DESARROLLO DE UNA RED SOCIAL MUSICAL PARA ANDROID

Trabajo final de grado

Estudiante:

Domínguez Purificación, Antonio

Directora:

Llorente Viejo, Silvia

28 de junio de 2015

Índice de figuras

1. Modelo conceptual de la primera iteración 37
2. Diagrama de casos de uso de la primera iteración 38
3. Modelo conceptual de la segunda iteración 45
4. Diagrama de casos de uso de la segunda iteración 46
5. Modelo conceptual de la tercera iteración 49
6. Diagrama de casos de uso de la tercera iteración 51
7. Primera parte del modelo conceptual de la cuarta iteración 62
8. Segunda parte del modelo conceptual de la cuarta iteración 63
9. Diagrama de casos de uso de la cuarta iteración 64
10. Modelo conceptual de la quinta iteración 68
11. Diagrama de casos de uso de la quinta iteración 69
12. Diagrama de casos de uso de la sexta iteración 73
13. Modelo conceptual completo 76
14. Porcentaje del uso de sistemas operativos en España, Alemania, E.E.U.U y México 79
15. Pantallas de login, registro y política de privacidad 92
16. Pantallas de creación de perfil y pantalla de elección de perfil al entrar en la aplicación 93
17. Pantalla de inicio: invitación de un grupo recibida, invitación de un evento recibida, menú desplegable 94
18. Pantalla para buscar un perfil, en este caso de artista, y resultados de la búsqueda 95
19. Mapa mostrando eventos cercanos hoy, pantalla de buscar evento y resultado de búsqueda 97
20. Pantalla de información de un evento 98
21. Pantallas de mensajes recibidos y de visualizar mensaje 99
22. Diálogo para escoger foto de perfil 100

23. Pantalla de perfil de fan 100
24. Pantalla de perfil de artista 102
25. Pantalla de perfil de grupo 103
26. Pantalla de perfil de local 104
27. Pantalla para cambiar entre perfiles de una cuenta 105
28. Pantalla de creación de perfiles, en este caso para cambiar de artista a fan 106
29. Pantallas de ajustes, modificar perfil y dialogo de confirmación para eliminar perfil 107
30. Pantalla que muestra las APIs habilitadas para el proyecto creado 108
31. Pantalla que muestra las claves para los servidores y aplicaciones Android especificadas 109
32. Diagrama del uso de Google Cloud Messaging 110
33. Diagrama de Gantt obtenido con la herramienta Gantter 122

Índice de tablas

1. Presupuesto de recursos humanos 124
2. Presupuesto de hardware 124
3. Presupuesto de software 125
4. Presupuesto total 125
5. Matriz de sostenibilidad 127

Índice de códigos

1. Código de ejemplo de como definir un modelo y sus relaciones en Flask-SQLAlchemy 83
2. Código de las funciones export_data e import_data 85
3. Código de ejemplo de operaciones CRUD sobre un modelo 87
4. Código de ejemplo de operaciones de búsqueda sobre perfiles, fan y artista en este caso 89
5. Código de ejemplo una petición GET con AQuery 91
6. Código para añadir párametros a una URL para una petición GET 96
7. Comandos para conseguir la firma SHA1 de apk-debug y apk-release . . . 109
8. Código para definir un mapa en una pantalla de la aplicación Android . . 111
9. Código de ejemplo de como manipular el mapa de la API de Google . . . 112
10. Antes y después de la línea que hay que modificar en la aplicación 112
11. Comando para entrar a la máquina virtual de Amazon EC2 a través de SSH 113
12. Contenido del fichero serverAPI.wsgi 114
13. Contenido del fichero amazonaws.com.conf 114
14. Comandos para activar un servicio y desactivar otro 114

Índice

1. Introducción y estado del arte **11**

1.1. Contexto 11

1.1.1. Glosario 11

1.1.2. Actores implicados 15

1.2. Estado del arte 16

1.2.1. Estudio de tecnologías 16

1.2.2. Servicio similares 17

1.2.3. Conclusiones 19

2. Alcance del proyecto y metodología **20**

2.1. Formulación del problema 20

2.2. Alcance 20

2.2.1. Posibles obstáculos 21

2.3. Metodología y rigor 22

2.3.1. Método de trabajo 22

2.3.2. Herramientas de seguimiento 22

2.3.3. Método de validación 23

3. Requisitos **24**

3.1. Requisitos funcionales 24

3.2. Requisitos de usabilidad y humanidad 34

3.3. Requisitos de rendimiento y escalabilidad 34

3.4. Requisitos políticos y/o culturales 35

4. Especificación **36**

4.1. Iteración 1: Cuentas, perfiles, asociados a cada cuenta y autentificación . . 37

4.1.1. Modelo conceptual 37

4.1.2. Casos de uso 38

4.2. Iteración 2: Sistema de seguimiento de perfiles 45
4.2.1. Modelo conceptual 45
4.2.2. Casos de uso 46
4.3. Iteración 3: Sistema de miembros por grupos, instrumentos y eventos . . . 49
4.3.1. Modelo conceptual 49
4.3.2. Casos de uso 51
4.4. Iteración 4: Sistema de publicaciones y mensajes entre perfiles 62
4.4.1. Modelo conceptual 62
4.4.2. Casos de uso 64
4.5. Iteración 5: Sistema de galería multimedia de cada perfil 68
4.5.1. Modelo conceptual 68
4.5.2. Casos de uso 69
4.6. Iteración 6: Sistema de búsqueda de perfiles y eventos 73
4.6.1. Casos de uso 73
4.7. Modelo conceptual completo 76

5. Diseño **78**
5.1. Tecnología 78
5.1.1. Servidor 78
5.1.2. Cliente 79
5.2. Arquitectura 80
5.2.1. Modelo 80
5.2.2. Vista 80
5.2.3. Controlador 81

6. Implementación **82**
6.1. Servidor 82
6.1.1. Modelo 82
6.1.2. Controlador 85

6.2. Cliente 90
6.2.1. Vista 90
6.3. APIs externas 107
6.3.1. *Google Cloud Messaging for Android* 110
6.3.2. *Google Maps Android API v2* 111
6.4. Despliegue 112

7. Planificación temporal **115**
7.1. Planificación general 115
7.1.1. Duración del proyecto 115
7.1.2. Recursos 115
7.1.3. Plan de acción 115
7.1.4. Análisis de alternativas 116
7.2. Descripción de las tareas 118
7.2.1. Investigación y toma de decisiones globales 118
7.2.2. Desarrollo del servidor 118
7.2.3. Desarrollo del cliente 119
7.3. Cambios desde la planificación inicial 120
7.4. Diagrama de Gantt 122

8. Gestión económica **123**
8.1. Consideraciones iniciales 123
8.2. Identificación y estimación de los costes 123
8.2.1. Presupuesto de recursos humanos 123
8.2.2. Presupuesto de hardware 124
8.2.3. Presupuesto de software 124
8.2.4. Presupuesto de mantenimiento 125
8.2.5. Presupuesto total 125
8.3. Control de gestión 125

9. Sostenibilidad y compromiso social **127**

9.1. Sostenibilidad económica 127

9.2. Sostenibilidad social 127

9.3. Sostenibilidad ambiental 128

10. Conclusiones **129**

10.1. Logro de objetivos 129

10.2. Futuro trabajo 129

10.3. Valoración personal 130

11. Referencias **132**

El mundo de la música es un campo artístico que puede llegar a ser muy productivo y satisfactorio para las personas. Sin embargo, tocar un instrumento o cantar no es la única experiencia, llega un momento en el que se encuentra un deseo de conocer a otra gente y empezar un proyecto musical. Empezar en este mundo sin ayuda es complicado, tanto para darse a conocer como para conocer a otros. Es por ello que se desarrolla este proyecto, para ofrecer una herramienta de apoyo a músicos, grupos, propietarios de local y también de ocio para amantes de la música. Esta red social ofrece la capacidad de poder buscar a fans, artistas, grupos y locales a través de distintos parámetros además de poder crear perfiles no tan sólo para un fan o artista, sino también para grupos y locales. Por último, otra de las funcionalidades más importantes es la de que un propietario de un local puede crear un evento e invitar a grupos. Gracias a estos eventos, cualquier usuario de la aplicación podrá acceder a un mapa y a un calendario con los eventos programados.

El món de la música és un camp artístic que pot arribar a ser molt productiu i satisfactori per a les persones. No obstant això, tocar un instrument o cantar no és l'única experiència, arriba un moment en què es troba un desig de conèixer altra gent i començar un projecte musical. Començar en aquest món sense ajuda és complicat, tant per donar-se a conèixer com per conèixer a altres. És per això que es desenvolupa aquest projecte, per oferir una eina de suport a músics, grups, propietaris de local i també d'oci per a amants de la música. Aquesta xarxa social ofereix la capacitat de poder buscar a fans, artistes, grups i locals a través de diferents paràmetres a més de poder crear perfils no tan sols per a un fan o artista, sinó també per a grups i locals. Finalment, una altra de les funcionalitats més importants és la de que un propietari d'un local pot crear un esdeveniment i convidar a grups. Gràcies a aquests esdeveniments, qualsevol usuari de l'aplicació podrà accedir a un mapa i a un calendari amb els esdeveniments programats.

The world of music is an artistic field that can be very productive and fulfilling for people. However, playing an instrument or singing is not the only experience, there comes a time when there is a desire to know other people and start a musical project. Starting in this world without help is difficult, to be known and to meet others. That is why this project is developed, to provide a support tool for musicians, groups, owners and local entertainment for music lovers. This social network provides the ability to search for fans, artists and properties through various parameters besides being able to create profiles not only for fans or artists, but also for groups and properties. Finally, one of the most important features is that an owner of a local can create an event and invite groups. Thanks to these events, any user of the application can access a map and a calendar of scheduled events.

1. Introducción y estado del arte

1.1. Contexto

Este proyecto se realiza como Trabajo Final del Grado (TFG) de Ingeniería Informática, especialidad de Tecnologías de la Información, en la Facultad de Informática de Barcelona, Universidad Politécnica de Cataluña (UPC).

El objetivo de este trabajo es el de ofrecer un servicio al sector de la música y unificar en una misma red social en Android a fans, artistas, bandas musicales y a dueños que tengan algún local en propiedad. La utilización de este servicio se justifica por el beneficio propio de sus usuarios, es decir, los artistas ingresarán y podrán conocer a otros artistas con los cuales poder formar una banda o ingresar en una banda ya creada, de la misma manera dueños de propiedades y bandas se podrán poner en contacto y organizar eventos.

Gracias a estos eventos todos los usuarios de la aplicación salen beneficiados:

- Artistas por haber encontrado grupo(s)
- Dueños de locales que obtendrán beneficio económico gracias al aumento de clientes
- Grupos que pueden darse a conocer entre más fans
- Fans que tendrán una agenda de eventos a los que poder acudir

1.1.1. Glosario

A continuación definiremos algunos de los componentes esenciales para entender el proyecto.

- **API (*Application Programming Interface*)** [1]: Una API es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

- **Base de datos relacional** [**2**]: Una base de datos relacional, es una base de datos que cumple con el modelo relacional, el cual es el modelo más utilizado en la actualidad para implementar bases de datos ya planificadas. Permiten establecer interconexiones (relaciones) entre los datos (que están guardados en tablas), y a través de dichas conexiones relacionar los datos de ambas tablas.

- **Cliente** [**3**]: El cliente es una aplicación informática o un ordenador que consume un servicio remoto en otro ordenador conocido como servidor, normalmente a través de una red de telecomunicaciones.

  En este caso es la aplicación Android encargada de parsear los mensajes JSON recibidos por el servidor y mostrarlo de una forma agradable y bonita para el usuario. El diseño de la aplicación es algo muy importante, ya que al ser una aplicación móvil tenemos una pantalla con el espacio muy limitado. Además, su diseño, funcionalidad y experiencia de usuario serán factores claves para la decisión del usuario de utilizar o no la aplicación.

- **Flask** [**4**]: Flask es un framework minimalista escrito en Python y basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2.

- ***Framework*** [**5**]: Un framework es una estructura de soporte definido, normalmente con artefactos o módulos de software concretos. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

- **Hipermedia** [**6**]: Hipermedia es el término con el que se designa al conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como: texto, imagen, video, audio, mapas y otros soportes de información emergentes, de tal modo que el resultado obtenido, además, tenga la posibilidad de interactuar con los usuarios.

- **HTTP (*Hypertext Transfer Protocol*)** [**7**]: HTTP es el protocolo usado en

cada transacción de la World Wide Web. Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente.

- **IDE (Integrated Development Environment)** [**8**]: Un IDE es una aplicación de software, que proporciona servicios integrales para facilitarle al programador de computadora el desarrollo de software. Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDEs tienen auto-completado inteligente de código.

- **Jinja2** [**9**]: Jinja2 es un *template engine* para Python.

- **JSON (*JavaScript Object Notation*)** [**10**]: JSON es un formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML (eXtensible Markup Language).

- **MySQL** [**11**]: MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

- **Perfil de artista**: Además de las funcionalidades de fan descritas anteriormente, el artista también tiene la capacidad de indicar su(s) instrumento(s), géneros, pequeña biografía, añadir videos de YouTube, asociar una cuenta de SoundCloud, la capacidad de crear un grupo y la capacidad de enviar mensajes a otros artistas y grupos.

- **Perfil de banda**: El perfil de banda tan sólo puede ser creado desde un perfil de artista y, una vez creado, el artista creador puede invitar a otros artistas en él. No tiene capacidad de seguir a otros perfiles puesto a que su uso es compartido. La funcionalidad de este perfil es la de reunir a los componentes del grupo, darse a conocer como tal, subir sus vídeos de YouTube y asociar un perfil de SoundCloud además de poder contactar con dueños de salas y viceversa.

- **Perfil de dueño de local**: Este perfil se asocia a unas coordenadas para indicar la posición de dicho local. Dichas coordenadas se utilizarán para la localización de eventos realizados en estos locales tanto como la localización del mismo. Los dueños de locales podrán crear eventos e invitar a grupos a participar en ellos. Al igual que los grupos, tampoco tienen la capacidad de seguir a ningún otro perfil ya que más de una persona puede estar administrando este tipo de perfil.

- **Perfil de fan**: La única funcionalidad de este perfil es la de poder seguir artistas, bandas, otros fans, locales, publicar texto y acceder al calendario de eventos. El objetivo de poder seguir perfiles es el de obtener un *feed* de noticias.

- **Python [12]**: Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

- **REST (*Representational State Transfer*) [13]**: REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

- **Servidor web [14]**: Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones. El término también se emplea para referirse al ordenador que ejecuta el programa.

  En este caso el servidor está formado únicamente por una API. Esta se comunica con el cliente a través de mensajes en formato JSON. La API ofrece capacidad de

gestionar perfiles, eventos y un sistema de búsqueda. Cabe destacar la utilización de una base de datos relacional, en este caso MySQL.

- **SoundCloud** [**15**]: SoundCloud es la plataforma social de música líder en el mundo en la que cualquier persona puede subir música y compartirla.
- ***Template engine*** [**16**]: Un *template engine* es un componente software diseñado para combinar una o más plantillas con modelos de datos y producir uno o más documentos resultados.
- **Werkzeug** [**17**]: Werkzeug es una librería WSGI de utilidades para Python.
- **WSGI (*Web Server Gateway Interface*)** [**18**]: WSGI es una espeficación para una interfaz simple y universal entre servidores web y aplicaciones cliente o frameworks para Python.

1.1.2. Actores implicados

Ahora detallaremos los actores implicados en el proyecto, es decir, las personas y organizaciones que pueden estar interesadas en el proyecto.

- **Desarrollador, beta-tester y diseñador**: Estos 3 roles son llevados a cabo por mí, el único encargado de realizar el proyecto.
- **Directora del proyecto**: La directora en este proyecto es Silvia Llorente Viejo. Es la encargada de supervisar el proyecto y de ayudar, guiar y resolver dudas al estudiante.
- **Usuarios**: Consideraremos usuarios a todo aquél que se aproveche del producto final de este proyecto. En este caso, hay varios tipos de usuarios: fans, artistas, bandas y dueños de locales. Cada tipo de usuario utilizará la aplicación con un objetivo distinto, por eso mismo cada uno tendrá un tipo de perfil con unas ciertas funcionalidades y capacidades específicas.

1.2. Estado del arte

El estado del arte de este proyecto se dividirá en dos partes: un estudio de servicios similares al planteado en el proyecto y un estudio de tecnologías utilizadas para desarrollar servidores web.

1.2.1. Estudio de tecnologías

A la hora de desarrollar el cliente Android sólo hay una única opción: Java y escoger un IDE entre Android Studio o Eclipse. En el caso del servidor se puede elegir entre muchos lenguajes, entre los más populares están PHP (Hypertext Preprocessor) y ASP.NET, los cuales ya abarcan un 99 % de todos los servidores de Internet, seguidos de Java, ColdFusion, Ruby, Perl, Python, JavaScript y Erlang [19]. Cada uno de estos lenguajes tiene sus propios frameworks tal y como Node.js para JavaScript, Laravel para PHP y Flask, Django y Pyramid para Python [20].

Entre todos los lenguajes elegí Python por dos motivos, primero porque es un lenguaje de alto nivel cuya filosofía es hacer más en menos líneas, lo que permite al desarrollador centrarse en otros ámbitos del desarrollo y segundo, porque ya había trabajado con Java, PHP sufre de puntos débiles como que es antiguo, hay muchas malas costumbres y códigos por Internet, es muy fácil hacer mal código con este lenguaje y quería aprovechar el desarrollo del TFG para aprender un lenguaje más. Entre Python hay varios frameworks a escoger como Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, etc. Los más populares de todos estos son los tres primeros que he mencionado y empecé a leer comparativas entre ellos. Django es, sin duda, el framework de Python más popular y utilizado y cuenta con una gran comunidad. Tanto Django como Pyramid están enfocados para proyectos web muy grandes. Flask, en cambio, es un microframework que ofrece una gran flexibilidad gracias a los plug-ins que hay desarrollados los cuales permiten añadir muchas funcionalidades [21].

Para acabar de tomar la decisión sobre que framework utilizar elegí por documentarme

sobre como crear una API con cada uno de estos tres frameworks y crear una implementación sencilla de operaciones CRUD (CREATE, READ, UPDATE, DELETE) sobre una misma tabla de una base de datos. Después de haber desarrollado esta pequeña API quedé sorprendido por la simplicidad de Flask y me decanté por este framework. Además, encontré un libro [22] de O'Reilly [23] sobre desarrollo web en Flask y una serie de videos [24] realizados por Miguel Grinberg [25] explicando como hacer una API con este framework.

1.2.2. Servicio similares

Actualmente existen alternativas de servicios similares al planteado en este proyecto. Sin embargo, hay pocas que sean españolas y todas son páginas web, todavía no hay ningún servicio similar en Android. A continuación haré un breve análisis junto con sus puntos débiles y fuertes de servicios que he encontrado.

- **Busco Grupo [26]**: Empezamos analizando Busco grupo, esta página web nos permite crear una cuenta y, posteriormente, un perfil a escoger entre músico, grupo, local de ensayo, profesor y vendedor de instrumentos. Una vez creado el perfil podemos anunciarnos y esperar a recibir mensajes con ofertas. La página web está bastante bien organizada pero algo que a mí no me gusta es que los mensajes están orientados entre cuentas, no entre perfiles. Esto último no es un inconveniente teniendo en cuenta que el propósito de esta página web es únicamente el de anunciarte.

- **Bandmix [27]**: Esta página web es, para mi gusto, la más completa. Además de estar disponible en varios países (España, Reino Unido, Francia, Australia, Canada) también tiene un diseño profesional que a simple vista, da confianza. Esta web permite crear un perfil como músico, banda o como industria (incluyendo fotógrafo, sala de ensayo, profesor, técnico de iluminación, etc). Una vez creado tu perfil cualquier persona puede encontrarte y contactar contigo. Además, cada perfil tiene un reproductor incrustado en el cual poder oír una muestra del artista

o banda en cuestión.

- **Local Rock [28]**: Con esta web se acaban los servicios destinados a España. En esta página web podemos crearnos una cuenta y, seguidamente, insertar los anuncios que queramos. Una vez puesto el anuncio cualquier persona puede contactar contigo a través del correo que has especificado a la hora del registro. Esta página web, además de tener un diseño poco atractivo no acaba de ser funcional. Por ejemplo, si eliges la opción buscar miembro e indicas que buscas cantante, en vez de salir como resultado los anuncios que cantantes que busquen grupos salen también anuncios de gente que también está buscando cantante. Esto hace que la búsqueda de lo que realmente quieres sea tediosa y poco eficiente.

- **Join My Band [29]**: Esta página web solo está destinada para Reino Unido. Su uso es solamente para insertar anuncios: te creas un perfil y creas un anuncio. No hay ningún sistema de clasificación de anuncios excepto por localización.

- **Band Finder [30]**: Seguimos con las páginas webs extranjeras. En este caso esta web sólo está disponible para USA, Reino Unido, Suecia y Canadá. En el momento del registro indicas si eres una banda buscando un músico, banda buscando fans, músico buscando banda, músico buscando a músico o músico buscando fans. Una vez creado tu perfil dispones de un reproductor multimedia en el cual poder insertar audio y cualquier usuario registrado puede contactar contigo a través de mensajes internos a la página web.

- **Find A Musician [31]**: Para acabar, esta página web sólo dispone de servicio en Australia, Nueva Zelanda, Canadá, USA, Reino Unido, Irlanda y Sud África. Una vez creado el perfil tienes la capacidad de colgar un anuncio, los anuncios se clasifican en disponible o buscado, es decir, si te ofreces como músico o si buscas a un músico. La página no tiene un diseño agradable, es un poco anticuado.

1.2.3. Conclusiones

Tal y como hemos visto, cada uno de estos servicios ofrece una serie de funcionalidades y experiencia de usuario distintas. Pero algo que comparten todas ellas es que están destinadas al uso como si de un tablón de anuncios se tratase.

Este proyecto sigue otra filosofía, la cual es hacer que el uso de la aplicación se prolongue a lo largo de la carrera musical de un artista o banda, que no sólo se use para insertar un anuncio pidiendo lo que buscas. Por este mismo motivo no podemos aprovechar ni hacer servir de modelo de ninguna de las soluciones ya existentes y hemos de estudiar un nuevo modelo de red social aplicado a música, bandas, artistas, fans y eventos.

Por último, cada día hay más gente que prefiere usar el Smartphone en cambio de un ordenador. Es decir, hay gente que el uso que le da a los ordenadores ya ha sido cubierto por la capacidad de los Smartphone y además con la libertad de poder hacerlo en cualquier lugar sin tener que estar atado a una silla, teclado y pantalla. Por esta razón y porque aún no hay ningún servicio similar al planteado en Android he decidido que era buena idea realizarlo.

2. Alcance del proyecto y metodología

2.1. Formulación del problema

Muchos músicos tienen problemas al buscar un grupo en el cual poder ingresar, ya sea por falta de contactos o porque no saben donde anunciarse. Lo mismo ocurre con grupos que no encuentran los miembros restantes para acabar de completarse. Para ello propongo esta red social, la cual será un lugar donde anunciarse, poder buscar oportunidades en cuanto al mundo de la música y que sirve como agenda de eventos musicales.

El principal objetivo de este proyecto es el de unificar en una misma red social a los amantes de la música, artistas, grupos y dueños de locales musicales con el propósito de facilitar el contacto entre estos colectivos. Gracias a esta red social, los artistas que quieran formar parte de un grupo musical pueden ponerse en contacto con otros artistas y formar uno o bien ingresar en grupos los cuales aún estén buscando miembros. Por otro lado, los fans pueden formar parte de esta red social y seguir a sus artistas y grupos favoritos. De este modo, los dueños de locales musicales pueden buscar grupos, viendo cuales son más populares e invitarlos a participar en eventos que ellos mismos pueden crear.

Otro objetivo de esta red social es el de que sirva como agenda para ver los eventos que tendrán lugar en los próximos días en una ubicación que el usuario indique. Además de poder acceder a un mapa que muestre los eventos que tengan lugar el mismo día de consulta y cercanos a la ubicación actual del usuario consultada vía GPS (*Global Positioning System*).

2.2. Alcance

El proyecto se divide en dos partes bien diferenciadas: el servidor y el cliente. El servidor se encarga de responder a las peticiones que se ejecuten sobre las URL's (*Uniform Resource Locator*) de la API. Esta API ofrece capacidad para gestionar los usuarios,

los tipos de perfiles de cada usuario, los eventos, los perfiles que sigue cada usuario, las notificaciones, solicitudes e invitaciones para participar en eventos y para ingresar en grupos y ofrecerá un sistema de búsqueda válido tanto para perfiles como para eventos gracias a parámetros como el municipio, nombre, género musical, fecha de evento, etc.

El cliente se basa en una aplicación Android la cual se comunica con el servidor a través de la API proporcionada y con mensajes HTTP en formato JSON. La aplicación ofrece una interfaz gráfica intuitiva, usable y con una experiencia de usuario agradable con la cual el usuario puede interactuar con la API del servidor. Por cada interacción entre el cliente y el servidor se efectua una petición HTTP en formato JSON y los parámetros correspondientes a la llamada efectuada. Posteriormente, el cliente Android recibe una respuesta en formato JSON y se obtiene la información deseada de esta respuesta para mostrarla al usuario de forma clara.

2.2.1. Posibles obstáculos

El primer obstáculo y el mayor que se plantea en este proyecto es el aprendizaje casi desde 0 en la gran mayoría del proyecto, tanto en la parte del servidor como en la aplicación Android. Se deberá aprender a utilizar el framework adecuado para desarrollar el servidor y su API, tal y como aprender a diseñar la estructura de una API REST de forma correcta. Por otro lado, aún habiendo desarrollado un par de aplicaciones Android aún hay muchas cosas que me quedan por aprender y que harán falta para cumplir todas las expectativas depositadas en el proyecto.

Los retrasos inesperados debidos a bugs, estancamientos, etc son otro posible obstáculo que hay que superar modificando la planificación temporal del proyecto, motivo por el cual debemos adoptar una cierta flexibilidad y planear la fecha prevista de finalización del proyecto tal que sea anterior a la de entrega.

2.3. Metodología y rigor

2.3.1. Método de trabajo

La metodología que se propuso para desarrollar el proyecto fue una metodología ágil y con iteraciones cortas. Esta metodología se basa en proponer unas metas cada semana, trabajar para conseguirlas y testear todas las funcionalidades desarrolladas esa semana. Destacar que esta metodología ha sido adaptada perfectamente y ha resultado bastante útil a la hora de controlar el tiempo e identificar las metas realizadas y las restantes. Por supuesto, el primer paso fue el diseño de la base de datos y el desarrollo del servidor tal y como su API.

Posteriormente se desarrolló la aplicación cliente, siguiendo el mismo patrón de desarrollo. En cada iteración se añadieron una serie de funcionalidades que, al terminar, se testeaban. Una vez finalizado el primer prototipo de aplicación fui yo mismo quien valoró si la interfaz era agradable, intuitiva y con una experiencia de usuario correcta. Se realizaron varias iteraciones sobre el diseño y sobre sus funcionalidades hasta que el resultado del proyecto fue satisfactorio. Finalmente se dipone la aplicación a los usuarios para que puedan opinar sobre ella y asi aplicar las iteraciones que sean necesarias hasta acabar de mejorar la aplicación.

En la sección del documento sobre la planificación se detallan de que tareas estaba formada cada parte del desarrollo del proyecto y una descripción de cada una.

2.3.2. Herramientas de seguimiento

Se ha utilizado Git y Github con el propósito de compartir el código entre distintas maquinas a la hora de desarrollar tanto el código del servidor como el de la aplicación Android. Utilizar estas herramientas también permite poder volver a versiones anteriores del código, copia de seguridad y muchas funcionalidades más, como por ejemplo la de desplegar el código del servidor en la máquina virtual de Amazon con tal sólo el comando

clone de Git.

2.3.3. Método de validación

Para comprobar el correcto desarrollo del proyecto se acordaron unas reuniones con la directora al inicio del desarrollo, para asegurar que el diseño de la base de datos era el correcto. Sin embargo, estas reuniones no fueron los únicos actos comunicativos, siempre estuvo abierta la comunicación vía correo electrónico entre alumno y directora en caso de posibles dudas por parte del alumno.

Al ser un proyecto realizado individualmente y en el que el cliente del proyecto es el propio desarrollador no habrá que realizar ningún tipo de reunión con ningún cliente. Será el propio alumno el que valore si el proyecto está yendo por el buen camino y, en caso contrario, el responsable de buscar soluciones a ello.

3. Requisitos

En esta sección del documento se listan todos los requisitos del proyecto tanto funcionales como de usabilidad, entre otros.

3.1. Requisitos funcionales

Requisito: Registro de usuario

Descripción: El sistema ha de permitir al usuario poder crearse una cuenta. Este registro se realizará con un correo electrónico y una contraseña. Además, también se pedirá al usuario que acepte la política de privacidad de la aplicación.

Requisito: Login

Descripción: El usuario ha de poder entrar al sistema con su usuario y contraseña.

Requisito: Logout

Descripción: El usuario ha de poder salir al sistema en cualquier momento.

Requisito: Modificar cuenta

Descripción: El usuario ha de poder modificar tanto la contraseña como el correo electrónico si así lo desea.

Requisito: Eliminar cuenta

Descripción: La cuenta se ha de poder eliminar si el usuario no quiere seguir utilizando el servicio. También se eliminarán los perfiles asociados a la cuenta.

Requisito: Crear un perfil

Descripción: Una vez creada una cuenta, dar capacidad para crear perfiles desde ella. Desde una cuenta se podrá crear un perfil de fan o artista (una exclusivamente) y/o de propietario de local

---

Requisito: Crear perfil de grupo

Descripción: La aplicación ha de permitir a cualquier usuario con un perfil de artista el poder crear un perfil de grupo.

Requisito: Visualizar perfil

Descripción: El sistema ha de ofrecer la capacidad de visualizar toda la información refente a un perfil.

Requisito: Modificar perfil

Descripción: Se trate de un perfil de fan, artista, grupo o de propietario de local,

todos han de poder modificar sus datos: nombre, localización, instrumento(s), cuenta de SoundCloud, etc.

Requisito: Eliminar perfil

Descripción: Capacidad para que el usuario elimine su perfil, eliminando así su información personal, otros perfiles asociados a este, etc.

Requisito: Buscar perfil

Descripción: Capacidad para que el usuario pueda buscar cualquier tipo de perfil especificando cualquier tipo de información como localización, nombre, etc.

Requisito: Subir foto de perfil o evento

Descripción: Capacidad de poder subir una foto desde la galería de imagenes del dispositivo Android o capturar una foto al momento para establecerla como foto de perfil o como foto de evento.

Requisito: Eliminar foto de perfil o evento

Descripción: Ofrecer capacidad de poder eliminar la foto de perfil o de evento si en cualquier momento el usuario no quiere mostrar ninguna y desea que vuelva a establecerse la imagen por defecto del sistema.

Requisito: Escribir publicación

Descripción: El usuario ha de poder, desde cualquier tipo de perfil, escribir publicaciones, estas publicaciones serán visibles desde su propio perfil y desde la pantalla de inicio de sus seguidores.

Requisito: Editar publicación

Descripción: El sistema ha de permitir al usuario editar las publicaciones de las cuales es el autor. Esto no modifica la hora de publicación.

Requisito: Eliminar publicación

Descripción: El sistema ha de permitir al usuario eliminar las publicaciones de las cuales es el autor.

Requisito: Visualizar publicaciones

Descripción: Desde el perfil de cualquier usuario se podrán ver todas sus publicaciones.

Requisito: Añadir género a perfil de artista o grupo

Descripción: El sistema ha de permitir a cualquier usuario con perfil de artista o de

grupo la posibilidad de añadir los géneros que toque.

Requisito: Eliminar género a perfil de artista o grupo

Descripción: El sistema ha de permitir a un usuario con perfil de artista o grupo la posibilidad de eliminar cualquiera de los géneros especificados por él.

Requisito: Añadir instrumento a perfil de artista o componente buscado a perfil de grupo

Descripción: El sistema ha de permitir a cualquier usuario con perfil de artista la posibilidad de añadir los instrumentos que toque y con perfil de grupo a añadir que componentes está buscando.

Requisito: Eliminar instrumento a perfil de artista o componente buscado a perfil de grupo

Descripción: El sistema ha de permitir a un usuario con perfil de artista la posibilidad de eliminar cualquiera de los instrumentos especificados por él y con perfil de grupo a eliminar los componentes que está buscando.

Requisito: Añadir vídeo de YouTube

Descripción: Se ha de ofrecer la capacidad que, desde un perfil de grupo o de artista, se pueda añadir tantos vídeos como se deseen.

Requisito: Modificar vídeo de YouTube

Descripción: Se ha de ofrecer la capacidad que, desde un perfil de grupo o de artista, se pueda modificar cualquier vídeo de los añadidos por él. De esta forma se podrá modificar la URL del vídeo sin tener que especificar de nuevo el nombre en caso de error o al contrario, modificar el nombre especificado al vídeo sin tener que especificar de nuevo su URL.

Requisito: Eliminar vídeo de YouTube

Descripción: Se ha de ofrecer la capacidad que, desde un perfil de grupo o de artista, se puedan añadir tantos vídeos como se deseen.

Requisito: Invitar artista a grupo

Descripción: Se ha de ofrecer la capacidad de que cualquier usuario en un perfil de grupo pueda invitar a otro artista a unirse.

Requisito: Eliminar artista de grupo

Descripción: Se ha de ofrecer la capacidad de que cualquier usuario en un perfil de grupo pueda eliminar a cualquier miembro del grupo, incluido él mismo, salvo que sea el último miembro. En este último caso se deberá de eliminar el grupo.

Requisito: Visualizar artistas de un grupo

Descripción: A través de la pantalla de perfil de grupo se puede ver los artistas que hay en él.

Requisito: Visualizar grupos de un artista

Descripción: A través de la pantalla de perfil de artista se puede ver los grupos en los que es miembro.

Requisito: Aceptar/rechazar invitación de grupo

Descripción: Se ha de ofrecer la capacidad de que cualquier usuario en un perfil de artista que haya recibido una invitación para ingresar a un grupo pueda aceptarla en caso de haberlo pactado así con el grupo o rechazarla en caso de que sea *spam*.

Requisito: Crear evento

Descripción: Desde un perfil de propietario de local se ha de poder crear un evento, cuya localización estará asociada a la ubicación del local.

Requisito: Editar evento

Descripción: Se ofrecerá la capacidad de que cualquier perfil de propietario pueda

modificar los datos de un evento crado previamente por él: nombre, descripción, hora y fecha.

Requisito: Eliminar evento

Descripción: Se debe de ofrecer la capacidad de eliminar un evento en caso de que el usuario asi lo desee ya sea en caso de evento cancelado o fallido o que el evento ya ha finalizado.

Requisito: Buscar evento

Descripción: Se debe de ofrecer la capacidad de que el usuario pueda buscar un evento por nombre, localización o simplemente que estén próximos a su localización GPS.

Requisito: Invitar grupo a evento

Descripción: Se debe de ofrecer la capacidad de eliminar un evento en caso de que el usuario asi lo desee ya sea en caso de evento cancelado o fallido o que el evento ya ha finalizado.

Requisito: Eliminar grupo de evento

Descripción: Se debe de ofrecer la capacidad de que el perfil de propietario que haya creado un evento pueda eliminar, si es necesario, un grupo que participa en dicho evento.

Requisito: Aceptar/rechazar invitación de evento

Descripción: Se debe de ofrecer la capacidad de que un usuario que haya sido invitado a participar en un evento pueda aceptar o rechazar dicha invitación.

Requisito: Visualizar grupos de un evento

Descripción: Desde la pantalla de descripción de evento se puede ver todos los grupos que participan en él.

Requisito: Visualizar eventos de un grupo

Descripción: Desde la pantalla de perfil de un grupo se puede ver todos los eventos que en los que participará próximamente.

Requisito: Seguir a un perfil

Descripción: Se ha de ofrecer la capacidad de que cualquier usuario con un perfil de fan o artista (estos dos perfiles son los únicos con esta capacidad) puedan seguir a cualquier perfil sea del tipo que sea.

Requisito: Dejar de seguir a un perfil

Descripción: Se ha de ofrecer la capacidad de que cualquier usuario con un perfil de

fan o artista puedan dejar seguir a cualquier perfil que ya siguieran anteriormente.

Requisito: Visualizar seguidores de un perfil

Descripción: Desde la página de perfil de cualquier usuario se podrá ver cuales son sus seguidores.

Requisito: Visualizar seguimientos de un perfil

Descripción: Desde la página de perfil de cualquier usuario se podrá ver cuales son los perfiles que él sigue.

Requisito: Escribir un mensaje

Descripción: Cualquier usuario con un perfil de artista, grupo o propietario de local puedan enviar mensajes a otros perfiles. Los fans no pueden enviar mensajes, los artistas sólo a otros artistas y a grupos, los grupos sólo a artistas, otros grupos y propietarios de local y estos últimos sólamente a grupos.

Requisito: Ver mensajes recibidos

Descripción: Cualquier usuario con un perfil de artista, grupo o propietario de local puedan ver los mensajes que han recibido.

3.2. Requisitos de usabilidad y humanidad

Requisito: Interfaz

Descripción: Interfaz simple e intuitiva con el objetivo de facilitar su uso al usuario.

Requisito: Datos actualizados

Descripción: El sistema ha de mostrar al usuario en todo momento los datos actualizados. Por ejemplo, si está visualizando sus publicaciones y elimina o edita una de ellas, este cambio ha de ser visible al momento, sin que sea necesario recargar la pantalla.

Requisito: Informar al usuario de errores

Descripción: El sistema ha de ser capaz de mostrar al usuario posibles errores que pueden ocurrir y permitir que siga utilizando la aplicación. Ejemplos de errores: Intentar publicar un mensaje vacío, añadir un vídeo sin URL, crear un perfil sin alguno de los campos obligatorios, etc.

3.3. Requisitos de rendimiento y escalabilidad

Requisito: Tiempo de respuesta

Descripción: El sistema ha de reaccionar lo más rápido posible a los cambios del usuarios.

Requisito: Escalabilidad

Descripción: El sistema ha de permitir que se pueda extender a demandas más altas.

Requisito: Extensibilidad

Descripción: El sistema ha de permitir que se puedan añadir más funciones según transcurre el tiempo..

3.4. Requisitos políticos y/o culturales

Requisito: Ley Orgánica de Protección de Datos de Carácter Personal de España (LOPD)

Descripción: El sistema ha de asegurar que la LOPD se cumple.

4. Especificación

Al haber empleado una metodología ágil en la que en cada semana se implementaban ciertas funciones, se especificará el modelo conceptual y los casos de uso por cada iteración realizada.

Al final de la sección se adjuntará el modelo conceptual completo.

4.1. Iteración 1: Cuentas, perfiles, asociados a cada cuenta y autentificación

4.1.1. Modelo conceptual

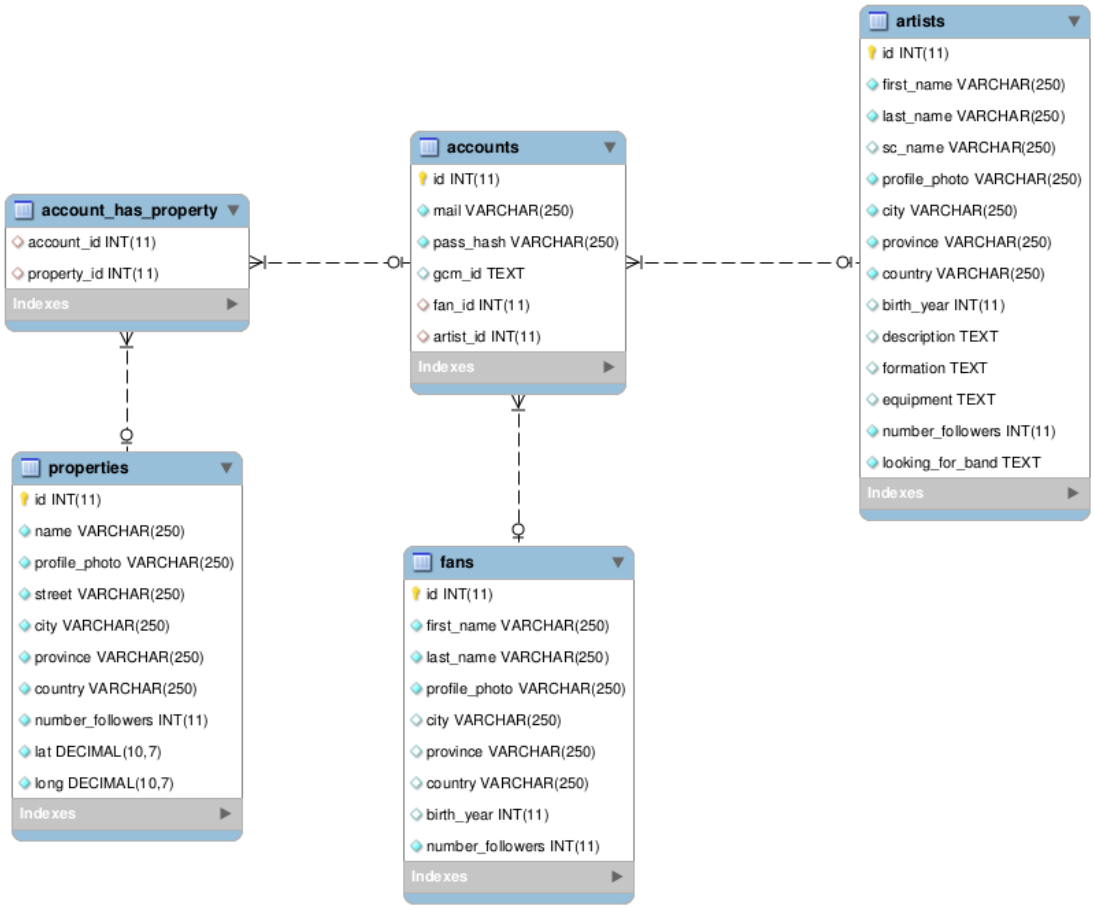

Figura 1: Modelo conceptual de la primera iteración

Restricciones

- Todas las clases han de tener un identificador
- El campo `mail` ha de ser único
- Para una cuenta sólo puede existir un artista o un fan, sólo uno exclusivamente

- Para cada `account` y `property` sólo puede existir una instancia de `account_has_property`

4.1.2. Casos de uso

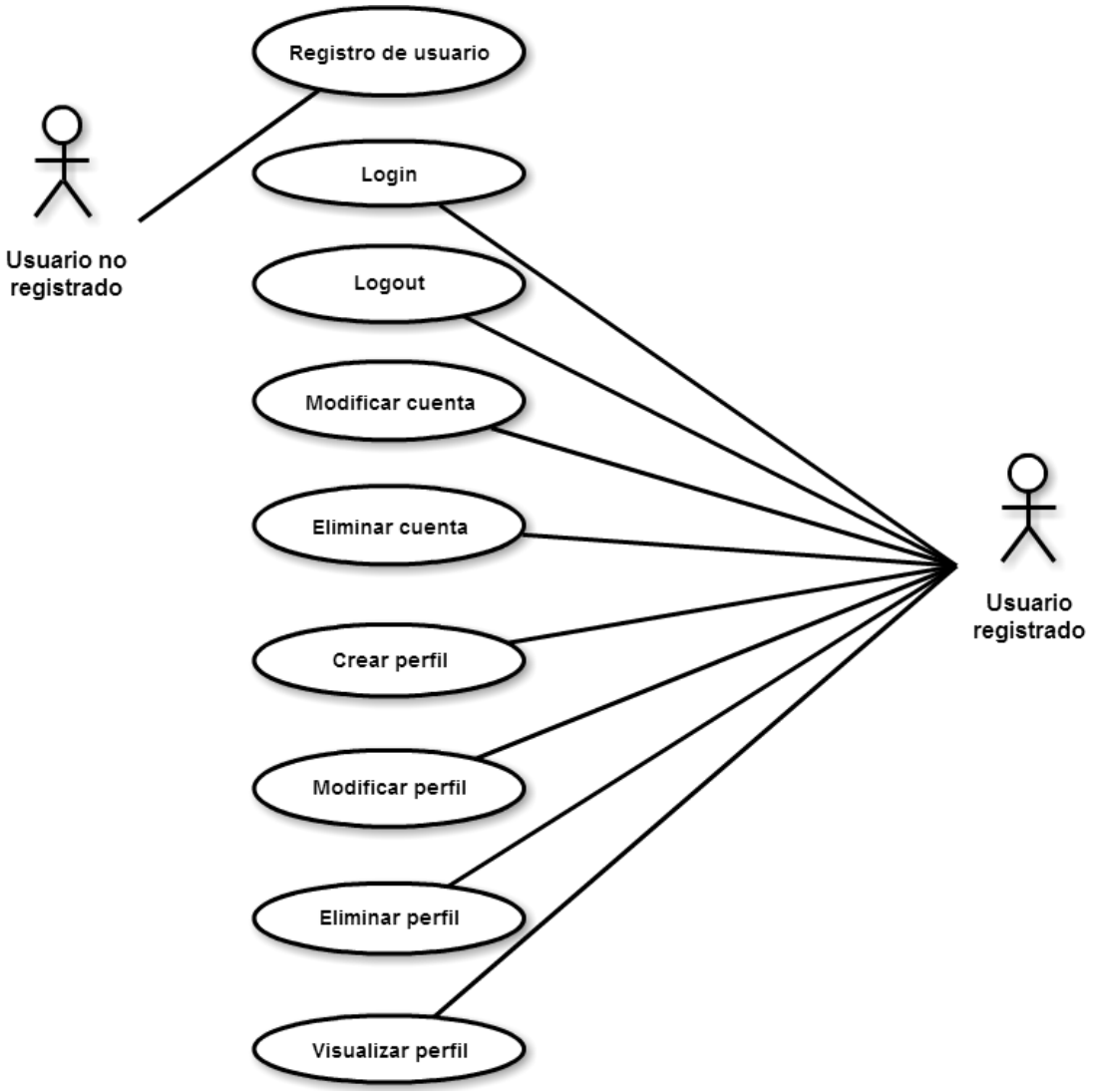

Figura 2: Diagrama de casos de uso de la primera iteración

Caso de uso: Registro de usuario

Actor: Usuario no registrado

Precondición: El usuario no registrado no tiene una cuenta en el sistema

Disparador: El usuario no registrado quiere entrar al sistema

Escenario principal:

1. El usuario entra en la pantalla de registro
2. El usuario completa el formulario con su correo electrónico, contraseña, confirmación de contraseña y acepta la política de privacidad
3. El usuario pulsa en el botón de registrarse
4. El servidor realiza un hash de la contraseña y lo guarda junto con el correo electrónico en la base de datos
5. El sistema lleva al usuario de vuelta a la pantalla de *login*

Caso de uso: *Login*

Actor: Usuario registrado

Precondición: El usuario tiene cuenta en el sistema

Disparador: El usuario registrado quiere entrar al sistema

Escenario principal:

1. El usuario entra en la pantalla de *login*
2. El usuario completa el formulario con su correo electrónico y contraseña
3. El usuario pulsa en el botón de *login*
4. El servidor realiza el hash de la contraseña introducida y verifica que sea igual al hash almacenado
5. En caso de que el paso anterior sea afirmativo, lleva al usuario a la pantalla principal de la aplicación

---

Caso de uso: *Logout*

Actor: Usuario registrado

Precondición: El usuario está dentro del sistema

Disparador: El usuario quiere volver a la pantalla de *login*

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación
2. El usuario selecciona la opción "cerrar sesión" del menú
3. El sistema cierra la pantalla principal de la aplicación
4. El sistema borra todos los datos referentes a la sesión actual
5. El sistema muestra la pantalla de *login*

Caso de uso: Modificar cuenta

Actor: Usuario registrado

Precondición: El usuario está dentro del sistema

Disparador: El usuario quiere modificar su correo electrónico y/o su contraseña

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación
2. El usuario selecciona la opción "ajustes" del menú
3. El usuario selecciona la opción "modificar cuenta" del menú de ajustes
4. El usuario completa el formulario mostrado por la aplicación: nuevo correo electrónico, confirmar nuevo correo (ambos campos por defecto completados con el correo actual), nueva contraseña, confirmar nueva contraseña, contraseña actual y el boton de modificar perfil

5. El sistema comprueba que los datos y sus confirmaciones coincidan, también que el hash de la contraseña introducida sea igual al almacenado

6. En caso afirmativo se modifican los datos en la base de datos y se vuelve a la pantalla anterior, en caso contario se informa del error y se permanece en la pantalla

---

Caso de uso: Eliminar cuenta

Actor: Usuario registrado

Precondición: El usuario está dentro del sistema

Disparador: El usuario quiere eliminar su contraseña porque no quiere seguir utilizando el servicio

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación

2. El usuario selecciona la opción "ajustes" del menú

3. El usuario selecciona la opción "eliminar cuenta" del menú de ajustes

4. El sistema muestra al usuario un diálogo pidiendo confirmación

5. Si el usuario confirma, el sistema borra la cuenta tal y como sus perfiles asociados y vuelve a la pantalla de *login*. En caso contrario todo permanece intacto

---

Caso de uso: Crear perfil

Actor: Usuario registrado

Precondición: El usuario está registrado

Disparador:

1. El usuario no tiene ningún perfil y entra al sistema por primera vez

2. El usuario quiere crear un perfil nuevo

Escenario principal:

- Si el usuario no tiene ninguún perfil y entra al sistema por primera vez
    1. El sistema muestra el formulario de crear perfil, el primer campo de este perfil es la selección de qué perfil se quiere crear
    2. El usuario completa el formulario y pulsa en el boton de "crear perfil"
    3. El sistema comprueba que los datos sean correctos
    4. En caso afirmativo lleva al usuario a la pantalla principal con su recién creado perfil
    5. En caso contrario se informa del error y se permanece en la pantalla de crear perfil
- El usuario quiere crear un perfil nuevo
    1. El usuario despliega el menú lateral de la aplicación y elige la opción "crear perfil"
    2. El sistema muestra al usuario el mismo formulario que en el caso anterior
    3. El usuario completa el formulario y pulsa en el boton de "crear perfil"
    4. El sistema comprueba que los datos sean correctos
    5. En caso afirmativo crea dicho perfil en la base de datos y lleva al usuario a la pantalla principal con su recién creado perfil
    6. En caso contrario se informa del error y se permanece en la pantalla de crear perfil

---

Caso de uso: Modificar perfil

Actor: Usuario registrado y con un perfil creado

Precondición: El usuario está dentro del sistema con uno de sus perfiles

Disparador: El usuario quiere modificar cualquier información de su perfil, sea del tipo que sea

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación
2. El usuario selecciona la opción "ajustes" del menú
3. El usuario selecciona la opción "modificar perfil" del menú de ajustes
4. El sistema muestra al usuario el mismo formulario que en el caso de crear perfil pero con los campos actuales ya completos
5. El usuario cambia los campos deseados y pulsa el boton de "modificar perfil"
6. EL sistema comprueba que todo sea correcto, almacena los cambios y se vuelve a la pantalla anterior o, en caso contrario, informa del error y se permanece en la pantalla actual

---

Caso de uso: Eliminar perfil

Actor: Usuario registrado y con perfil

Precondición: El usuario está dentro de la aplicación con el perfil que quiere eliminar

Disparador: El usuario quiere eliminar el perfil actual

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación
2. El usuario selecciona la opción "ajustes" del menú
3. El usuario selecciona la opción "eliminar perfil" del menú de ajustes
4. El sistema muestra al usuario un diálogo pidiendo confirmación

5. Si el usuario confirma, el sistema borra el perfil tal y vuelve a la pantalla de *login*. En caso contrario todo permanece intacto

---

Caso de uso: Visualizar perfil

Actor: Usuario registrado y con perfil

Precondición: El usuario está dentro de la aplicación con el perfil que quiere ver

Disparador: El usuario quiere ver los datos del perfil actual

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación
2. El usuario selecciona la opción "mi perfil" del menú
3. El sistema muestra al usuario la pantalla de perfil mostrando su información

4.2. Iteración 2: Sistema de seguimiento de perfiles

4.2.1. Modelo conceptual

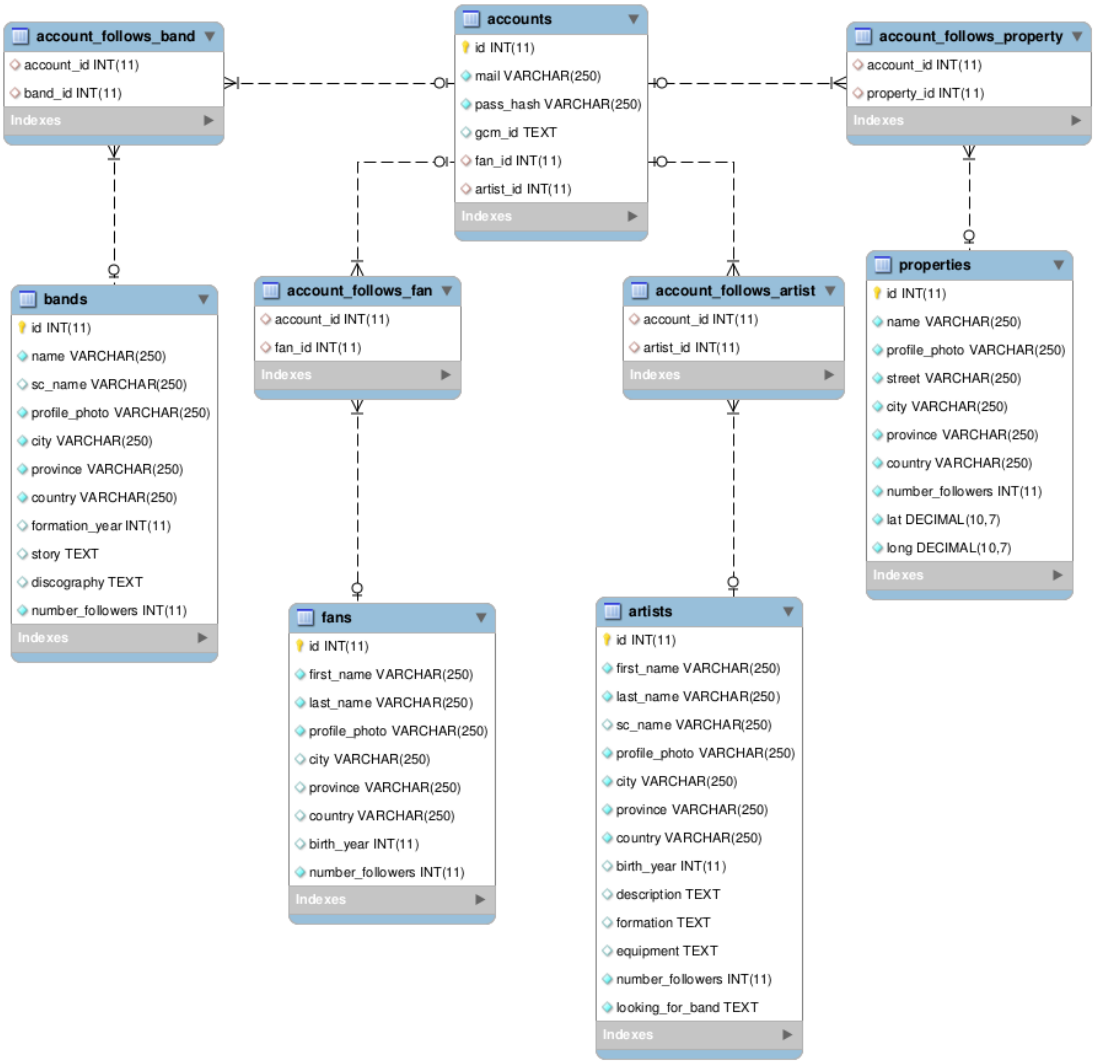

Figura 3: Modelo conceptual de la segunda iteración

Restricciones

- Todas las clases han de tener un identificador

- El campo `mail` ha de ser único
- Para cada `account` y `property` sólo puede existir una instancia de `account_follows_property`
- Para cada `account` y `band` sólo puede existir una instancia de `account_follows_band`
- Para cada `account` y `artist` sólo puede existir una instancia de `account_follows_artist`
- Para cada `account` y `fan` sólo puede existir una instancia de `account_follows_fan`

4.2.2. Casos de uso

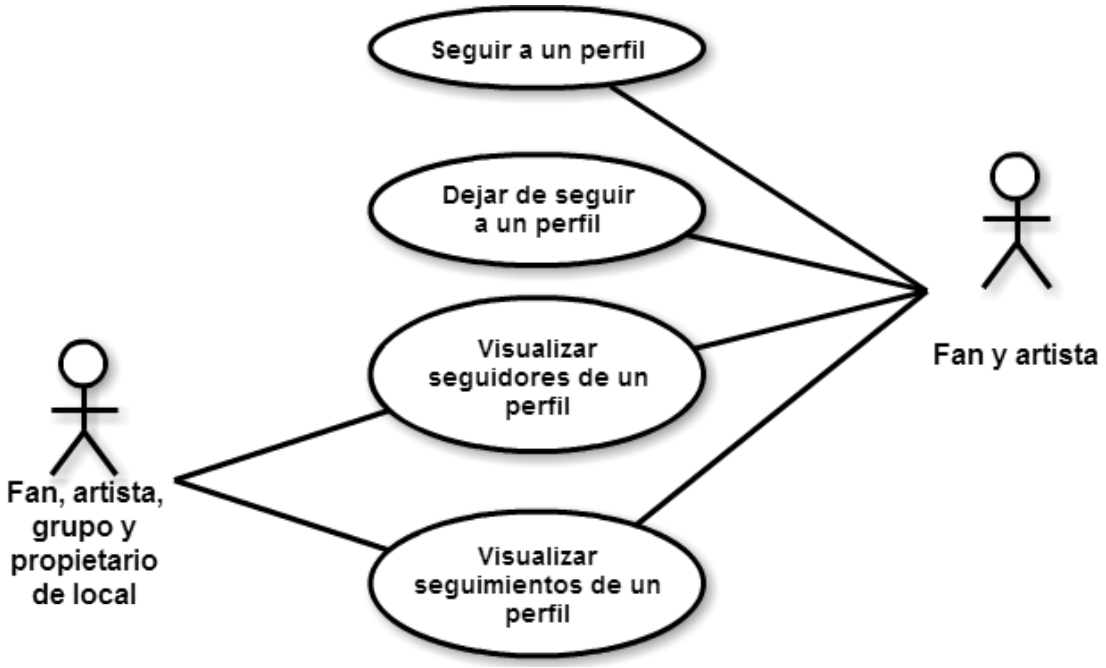

Figura 4: Diagrama de casos de uso de la segunda iteración

Caso de uso: Seguir a un perfil

Actor: Fan o artista

Precondición: Ambos perfiles existen y el que quiere seguir al otro es un fan o artista

Disparador: El fan/artista quiere recibir en su *feed* de la pantalla inicial las últimas noticias sobre el perfil al que quiere seguir

Escenario principal:

1. El fan/artista entra en el perfil que quiere seguir y pulsa el botón de seguir o pulsa dicho botón desde cualquier otra lista de cualquier otro perfil (seguidores, miembros de grupo, etc)

2. El sistema almacena en la base de datos la asociacion entre el identificador de la cuenta y del perfil

---

Caso de uso: Dejar de seguir a un perfil

Actor: Fan o artista

Precondición: Ambos perfiles existen, el que quiere seguir al otro es un fan o artista y ya lo sigue previamente

Disparador: El fan/artista no quiere quiere recibir de ahora en adelante en su *feed* de la pantalla inicial las últimas noticias sobre el perfil al que quiere dejar seguir

Escenario principal:

1. El fan/artista entra en el perfil que quiere seguir y pulsa el botón de dejar de seguir o pulsa dicho botón desde cualquier otra lista de cualquier otro perfil (seguidores, miembros de grupo, etc)

2. El sistema elimina de la base de datos la asociacion entre el identificador de la cuenta y del perfil ya creada

---

Caso de uso: Visualizar seguidores de un perfil

Actor: Cualquier perfil

Precondición: El perfil existe

Disparador: Se quiere observar qué seguidores tiene un perfil determinado
Escenario principal:

1. El usuario entra al perfil que quiere ver sus seguidores
2. El sistema muestra al usuario la pantalla de perfil y una serie de pestañas, entre ellas la de "seguidores"
3. El usuario navega hasta dicha pestaña
4. El sistema muestra los seguidores de dicho perfil

Caso de uso: Visualizar seguimientos de un perfil
Actor: Cualquier perfil
Precondición: El perfil existe
Disparador: Se quiere observar qué seguimientos tiene un perfil determinado
Escenario principal:

1. El usuario entra al perfil que quiere ver sus seguidores
2. El sistema muestra al usuario la pantalla de perfil y una serie de pestañas, entre ellas la de "seguidos"
3. El usuario navega hasta dicha pestaña
4. El sistema muestra los seguidos de dicho perfil

4.3. Iteración 3: Sistema de miembros por grupos, instrumentos y eventos

4.3.1. Modelo conceptual

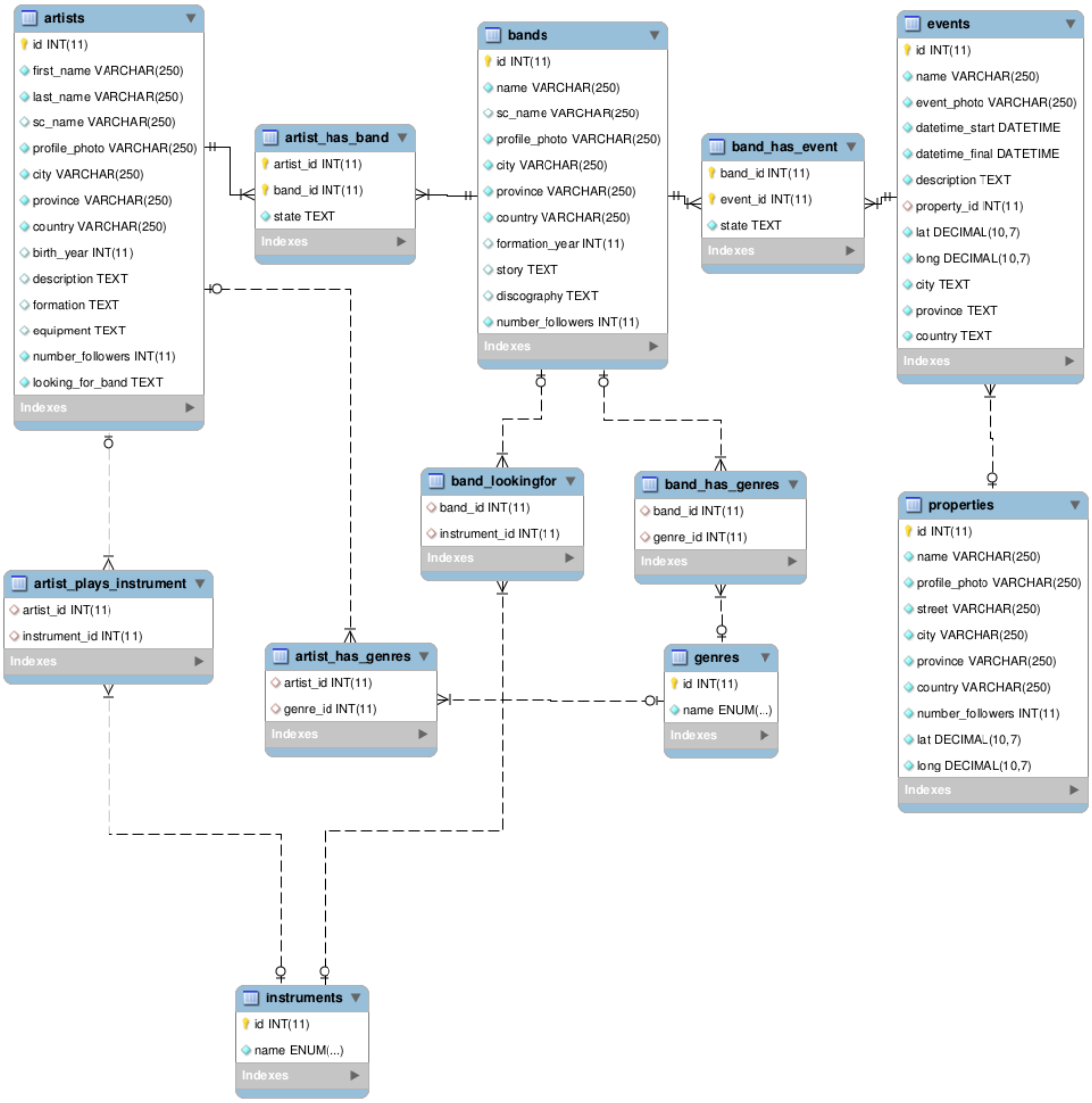

Figura 5: Modelo conceptual de la tercera iteración

Restricciones

- Todas las clases han de tener un identificador

- Para cada `band` y `artist` sólo puede existir una instancia de `artist_has_band`
- Para cada `band` y `event` sólo puede existir una instancia de `band_has_event`
- Para cada `band` y `genres` sólo puede existir una instancia de `band_has_genres`
- Para cada `artist` y `genres` sólo puede existir una instancia de `artist_has_genres`
- Para cada `band` y `instruments` sólo puede existir una instancia de `band_lookingfor`
- Para cada `artist` y `instruments` sólo puede existir una instancia de `artist_plays_instrument`

4.3.2. Casos de uso

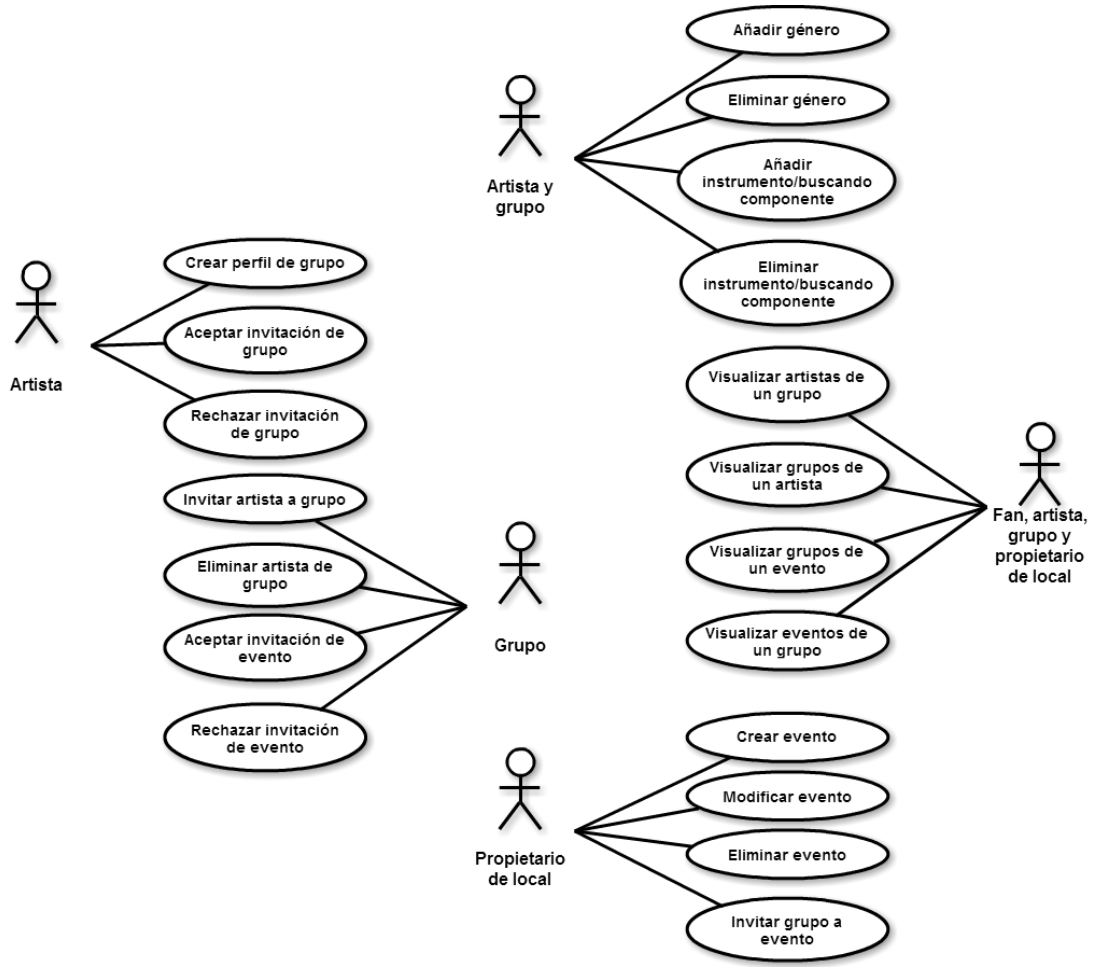

Figura 6: Diagrama de casos de uso de la tercera iteración

Caso de uso: Crear perfil de grupo

Actor: Artista

Precondición: El artista ha de existir

Disparador: El artista quiere crear un nuevo grupo

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación y elige la opción “crear perfil”
2. El sistema muestra al usuario un formulario
3. El usuario escoge crear un grupo en el formulario, lo completa y pulsa en el boton de “crear perfil”
4. El sistema comprueba que los datos sean correctos
5. En caso afirmativo se crea dicho perfil y se lleva al usuario a la pantalla principal con su recién creado perfil
6. En caso contrario se informa del error y se permanece en la pantalla de crear perfil

---

Caso de uso: Aceptar invitación de grupo
Actor: Artista
Precondición: Un grupo ha de haber invitado anteriormente al artista
Disparador: El artista quiere unirse al grupo al que le han invitado
Escenario principal:

1. El usuario ha de entrar en la pantalla principal, en esta pantalla se muestra la invitación y dos botones: aceptar y rechazar
2. El usuario pulsa sobre el boton de aceptar
3. El sistema cambia el estado de la asociación entre artista y grupo de *pending* a *active*

Caso de uso: Rechazar invitación de grupo
Actor: Artista
Precondición: Un grupo ha de haber invitado anteriormente al artista

Disparador: El artista no quiere unirse al grupo al que le han invitado

Escenario principal:

1. El usuario ha de entrar en la pantalla principal, en esta pantalla se muestra la invitación y dos botones: aceptar y rechazar
2. El usuario pulsa sobre el boton de rechazar
3. El sistema borra la asociación entre grupo y artista

Caso de uso: Invitar artista a grupo

Actor: Grupo

Precondición: Grupo y artista han de existir

Disparador: El grupo desea que el artista sea un componente

Escenario principal:

1. Cualquier componente actual del grupo desde dicho perfil visita el perfil del artista a invitar
2. El usuario pulsa sobre el botón de invitar a grupo
3. El sistema pide confirmación
4. En caso afirmativo se crea una asociación entre artista y grupo con estado *pending* y el servidor envía una petición al servidor de Google Cloud Messaing para que se notifique al usuario sobre dicha invitación

---

Caso de uso: Eliminar artista de grupo

Actor: Grupo

Precondición: El artista ha de ser component del grupo

Disparador: Uno de los artistas quiere dejar el grupo

Escenario principal:

1. Cualquier componente actual del grupo desde dicho perfil visita el perfil de grupo
2. El usuario navega a través de las pestañas hasta la de miembros
3. Mantiene pulsado el item de la lista correspondiente al artista que quiere eliminar
4. El sistema pide confirmación para eliminarlo
5. En caso afirmativo elimina la asociación entre artista y grupo, en caso contrario todo permanece intacto

---

Caso de uso: Aceptar invitación de evento
Actor: Grupo
Precondición: El grupo ha sido invitado a participar a un evento
Disparador: El grupo quiere unirse al evento que le han invitado
Escenario principal:

1. El usuario ha de entrar en la pantalla principal, en esta pantalla se muestra la invitación y dos botones: aceptar y rechazar
2. El usuario pulsa sobre el boton de aceptar
3. El sistema cambia el estado de la asociación entre grupo y evento de *pending* a *active*

Caso de uso: Rechazar invitación de evento
Actor: Grupo
Precondición: El grupo ha sido invitado a partcipar a un evento
Disparador: El grupo no quiere participar en el evento
Escenario principal:

1. El usuario ha de entrar en la pantalla principal, en esta pantalla se muestra la invitación y dos botones: aceptar y rechazar
2. El usuario pulsa sobre el boton de rechazar
3. El sistema borra la asociación entre grupo y evento

Caso de uso: Añadir género
Actor: Artista y grupo
Precondición: El artista/grupo existe y no ha superado el límite de géneros
Disparador: El artista/grupo quiere añadir un género a su perfil
Escenario principal:

1. El usuario ha de desplegar el menú lateral y pulsar sobre ajustes
2. El sistema muestra la pantalla de ajustes
3. El usuario escoge la opción de modificar perfil
4. El sistema muestra el formulario con todos los datos, entre ellos, géneros
5. El usuario añade los géneros que quiera y pulsa sobre modificar
6. El sistema almacena una asociación entre el artista/grupo y el género especificado

---

Caso de uso: Eliminar género
Actor: Artista y grupo
Precondición: El artista/grupo existe y tiene al menos un género
Disparador: El artista/grupo quiere eliminar un género de su perfil
Escenario principal:

1. El usuario ha de desplegar el menú lateral y pulsar sobre ajustes

2. El sistema muestra la pantalla de ajustes
3. El usuario escoge la opción de modificar perfil
4. El sistema muestra el formulario con todos los datos, entre ellos, géneros
5. El usuario elimina los géneros que quiera y pulsa sobre modificar
6. El sistema elimina la asociación entre el artista/grupo y el género eliminado

---

Caso de uso: Añadir instrumento/buscando componente

Actor: Artista y grupo

Precondición: El artista/grupo existe y no ha superado el límite de instrumentos/buscando componentes

Disparador: El artista/grupo quiere añadir un instrumento/buscando componente a su perfil

Escenario principal:

1. El usuario ha de desplegar el menú lateral y pulsar sobre ajustes
2. El sistema muestra la pantalla de ajustes
3. El usuario escoge la opción de modificar perfil
4. El sistema muestra el formulario con todos los datos, entre ellos, instrumentos/buscando componentes
5. El usuario añade los insrtumentos/buscando componentes que quiera y pulsa sobre modificar
6. El sistema almacena una asociación entre el artista/grupo y el instrumento especificado

---

Caso de uso: Eliminar instrumento/buscando componente

Actor: Artista y grupo

Precondición: El artista/grupo existe y tiene al menos un instrumento/buscando componente

Disparador: El artista/grupo quiere eliminar un instrumento/buscando componente de su perfil

Escenario principal:

1. El usuario ha de desplegar el menú lateral y pulsar sobre ajustes
2. El sistema muestra la pantalla de ajustes
3. El usuario escoge la opción de modificar perfil
4. El sistema muestra el formulario con todos los datos, entre ellos, instrumentos/buscando componentes
5. El usuario elimina los instrumentos/buscando componentes que quiera y pulsa sobre modificar
6. El sistema elimina la asociación entre el artista/grupo y el instrumento eliminado

---

Caso de uso: Visualizar artistas de un grupo

Actor: Cualquier perfil

Precondición: El grupo debe existir

Disparador: El usuario quiere ver que componentes forman un grupo

Escenario principal:

1. El usuario visita el perfil de grupo

2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario navega hasta la pestaña “miembros”
4. El sistema busca en la base de datos todos los miembros de ese grupo y los muestra

---

Caso de uso: Visualizar grupos de un artista
Actor: Cualquier perfil
Precondición: El artista debe existir
Disparador: El usuario quiere ver en qué grupos está un artista
Escenario principal:

1. El usuario visita el perfil de artista
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario navega hasta la pestaña “grupos”
4. El sistema busca en la base de datos todos los grupos de dicho artista y los muestra

---

Caso de uso: Visualizar grupos de un evento
Actor: Cualquier perfil
Precondición: El evento debe existir
Disparador: El usuario quiere ver que grupos participan en el evento
Escenario principal:

1. El usuario visita la pantalla del evento
2. El sistema muestra la pantalla del evento con información básica y unas pestañas con información más detallada

3. El usuario navega hasta la pestaña “grupos”
4. El sistema busca en la base de datos todos los grupos de ese evento y los muestra

---

Caso de uso: Visualizar eventos de un grupo
Actor: Cualquier perfil
Precondición: El grupo debe existir
Disparador: El usuario quiere ver en qué eventos participa un grupo
Escenario principal:

1. El usuario visita el perfil de grupo
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario navega hasta la pestaña “eventos”
4. El sistema busca en la base de datos todos los eventos en los que participa dicho grupo y los muestra

---

Caso de uso: Crear evento
Actor: Propietario de local
Precondición: El perfil de propietario debe de existir
Disparador: El propietario de local quiere crear un evento
Escenario principal:

1. El usuario visita su propio perfil
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada

3. El usuario navega hasta la pestaña “eventos” y pulsa sobre el botón crear evento
4. El sistema muestra un formulario con: nombre, descripción, fecha y hora de inicio y de final
5. El usuario completa el formulario y pulsa sobre crear evento
6. El sistema comprueba que los datos sean correctos. En caso afirmativo se crea en la base de datos el evento y la asociación entre evento y local

---

Caso de uso: Modificar evento
Actor: Propietario de local
Precondición: El evento debe de existir y el perfil de propietario ser su creador
Disparador: El propietario de local quiere modificar un evento creado
Escenario principal:

1. El usuario visita su propio perfil
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario navega hasta la pestaña “eventos” y, en el listado de eventos, pulsa el botón de modificar sobre el que desee
4. El sistema muestra un formulario con: nombre, descripción, fecha y hora de inicio y de final
5. El usuario completa el formulario y pulsa sobre modificar evento
6. El sistema comprueba que los datos sean correctos. En caso afirmativo se modifica en la base de datos el evento

---

Caso de uso: Eliminar evento

Actor: Propietario de local

Precondición: El evento debe de existir y el perfil de propietario ser su creador

Disparador: El propietario de local quiere eliminar un evento creado

Escenario principal:

1. El usuario visita su propio perfil
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario navega hasta la pestaña "eventos" y, en el listado de eventos, pulsa el botón de eliminar sobre el que desee
4. El sistema muestra una pantalla de confirmación
5. En caso afirmativo se elimina el evento y todas sus posibles asociacionse a grupos y al local, en caso contrario todo queda intacto

---

Caso de uso: Invitar grupo a un evento

Actor: Propietario de local

Precondición: El evento y grupo existen

Disparador: El propietario de local quiere invitar a un grupo a un evento previamente creado

Escenario principal:

1. El usuario visita el perfil de grupo
2. El sistema muestra la pantalla de perfil con información básica y unas pestañas con información más detallada
3. El usuario pulsa sobre el botón invitar a evento

4. El sistema muestra una pantalla con todos los eventos creado por el propietario pregutando a cual quiere invitar al grupo

5. El usuario pulsa sobre el evento

6. El sistema crea una asociación entre evento y grupo con estado *pending* y el servidor envía una petición al servidor de Google Cloud Messaing para que se notifique a los usuarios del grupo sobre dicha invitación

4.4. Iteración 4: Sistema de publicaciones y mensajes entre perfiles

4.4.1. Modelo conceptual

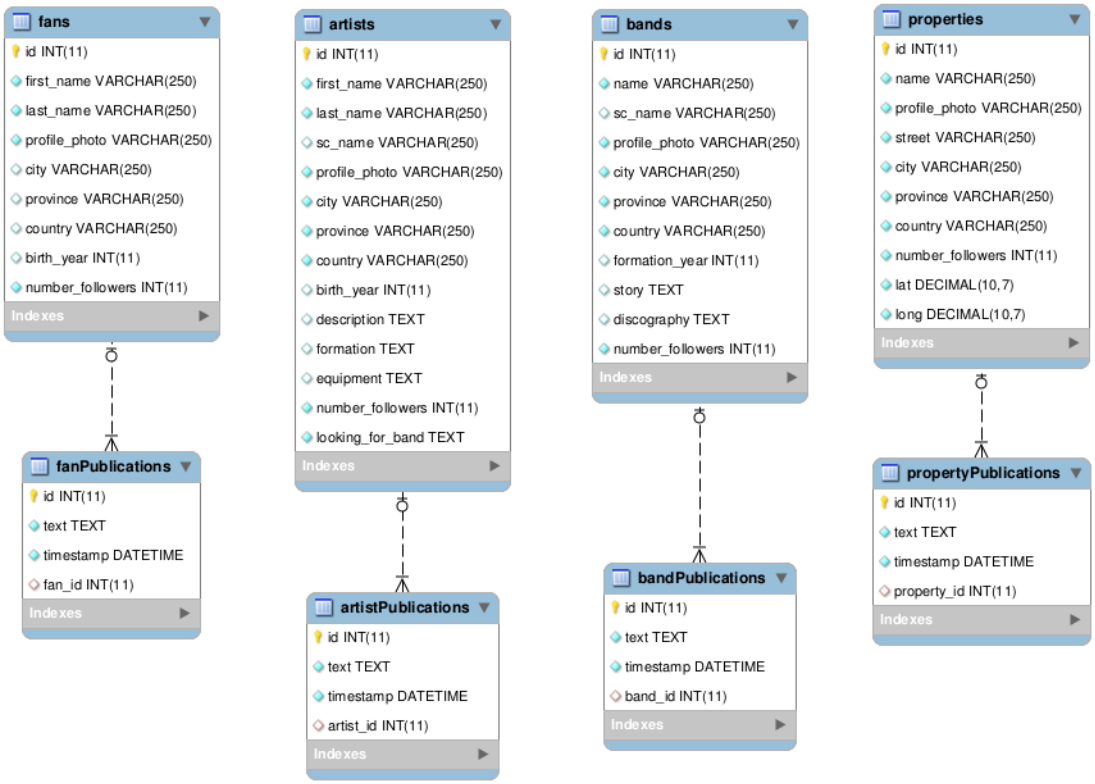

Figura 7: Primera parte del modelo conceptual de la cuarta iteración

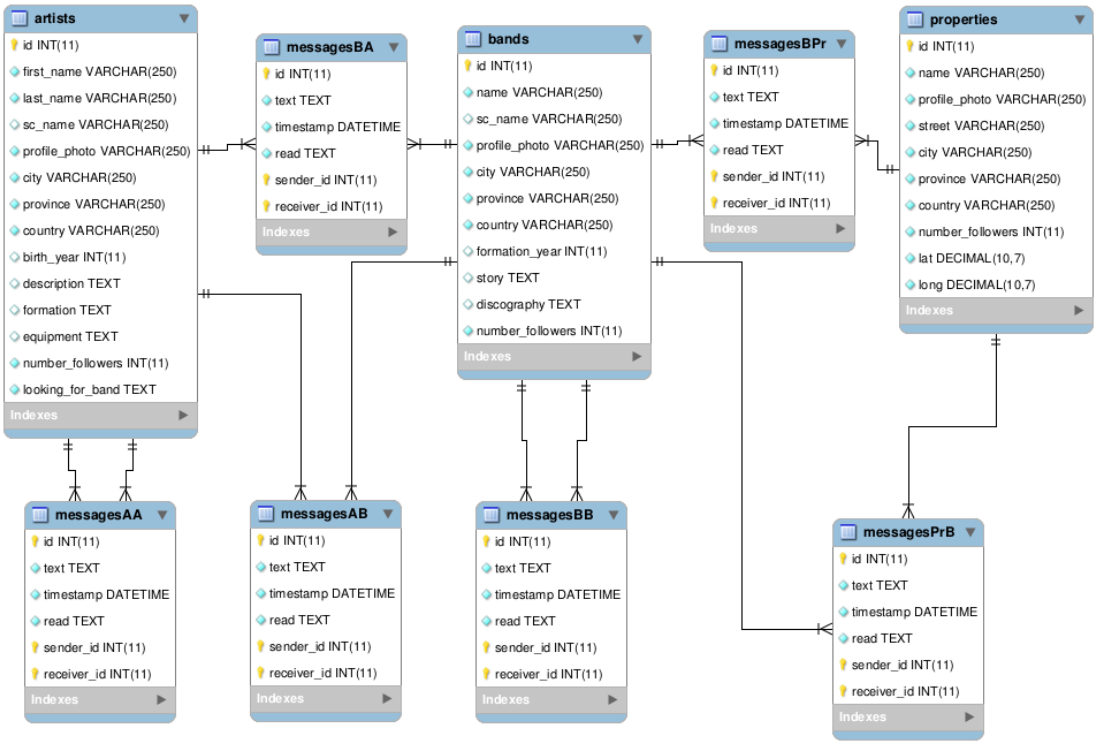

Figura 8: Segunda parte del modelo conceptual de la cuarta iteración

Restricciones

- Todas las clases han de tener un identificador
- El campo `timestamp` toma por defecto el instante de tiempo al crearse la instancia
- El campo `read` por defecto es false, este valor cambia cuando se abre el mensaje en la aplicación

4.4.2. Casos de uso

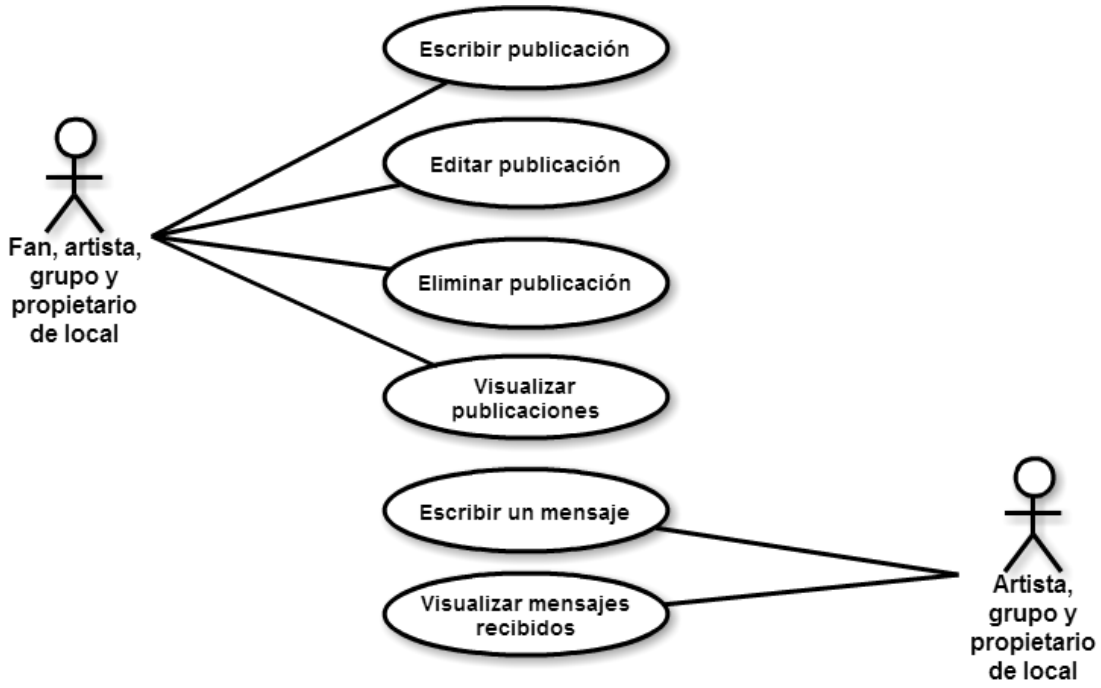

Figura 9: Diagrama de casos de uso de la cuarta iteración

Caso de uso: Escribir publicación

Actor: Cualquier perfil

Precondición: El perfil existe y el mensaje no es vacío

Disparador: El usuario quiere publicar un mensaje visible en su perfil y en el inicio de sus seguidores

Escenario principal:

1. El usuario se dirige a la pantalla inicial
2. El sistema muestra una pantalla con un campo para escribir la publicación
3. El usuario escribe el texto deseado y pulsa en publicar

4. El sistema almacena la publicación en la base de datos

Caso de uso: Editar publicación

Actor: Cualquier perfil

Precondición: Mensaje existe y el perfil es su autor

Disparador: El usuario quiere editar una publicación escrita por él anteriormente

Escenario principal:

1. El usuario se dirige a su perfil y, en él, a la pestaña publicaciones
2. El sistema muestra el listado de todas sus publicaciones
3. El usuario pulsa sobre el boton de editar en la publicación que desee
4. El sistema muestra el campo de texto con el texto actual
5. El usuario modifica el texto y pulsa en editar
6. El sistema almacena el cambio de la publicación en la base de datos

---

Caso de uso: Eliminar publicación

Actor: Cualquier perfil

Precondición: Mensaje existe y el perfil es su autor

Disparador: El usuario quiere eliminar una publicación escrita por él anteriormente

Escenario principal:

1. El usuario se dirige a su perfil y, en él, a la pestaña publicaciones
2. El sistema muestra el listado de todas sus publicaciones
3. El usuario pulsa sobre el boton de eliminar en la publicación que desee

4. El sistema pide confirmación
5. En caso afirmativo se elimina de la base de datos la publicación y en caso contrario todo queda intacto

---

Caso de uso: Visualizar publicaciones
Actor: Cualquier perfil
Precondición: El perfil existe
Disparador: El usuario quiere ver sus publicaciones, editarlas o borrarlas
Escenario principal:

1. El usuario se dirige a su perfil y, en él, a la pestaña publicaciones
2. El sistema muestra el listado de todas sus publicaciones

Caso de uso: Escribir mensaje
Actor: Artista, grupo y propietario de local
Precondición: Ambos perfiles existen
Disparador: El usuario quiere enviar un mensaje a otro
Escenario principal:

1. El usuario se dirige al perfil de la persona a la cual quiere enviar un mensaje y pulsa el botón de escribir mensaje o va a un mensaje recibido y pulsa sobre el botón de responder
2. El sistema muestra un campo de texto en el cual introducir el texto
3. El usuario pulsa sobre enviar
4. El sistema almacena el mensaje en la base de datos, relacionando ambos perfiles

Caso de uso: Visualizar mensajes

Actor: Artista, grupo y propietario de local

Precondición: El perfil existe

Disparador: El usuario quiere ver los mensajes recibidos

Escenario principal:

1. El usuario abre el menú lateral y pulsa sobre “mensajes”
2. El sistema muestra la pantalla con el listado de mensajes clasificados por el tipo de perfil que los ha enviado

4.5. Iteración 5: Sistema de galería multimedia de cada perfil

4.5.1. Modelo conceptual

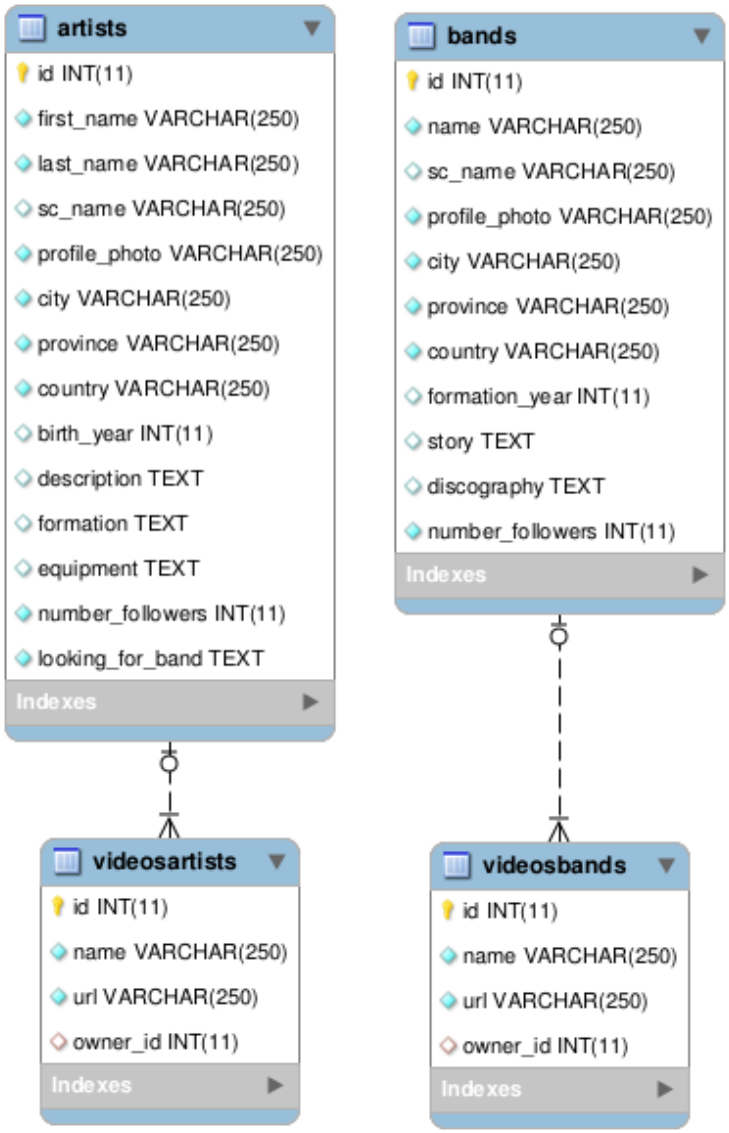

Figura 10: Modelo conceptual de la quinta iteración

Restricciones

- Todas las clases han de tener un identificador

4.5.2. Casos de uso

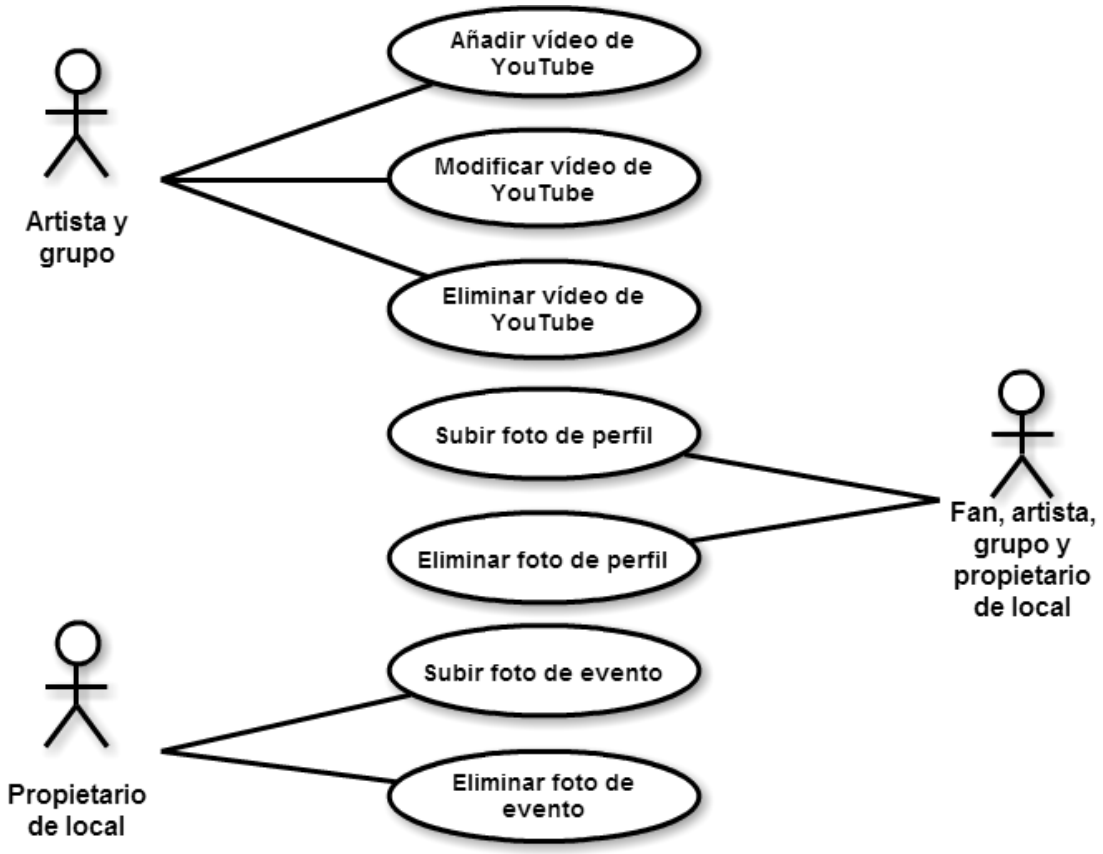

Figura 11: Diagrama de casos de uso de la quinta iteración

Caso de uso: Añadir vídeo de YouTube

Actor: Artista y grupo

Precondición: El perfil existe

Disparador: El usuario quiere añadir un vídeo de YouTube a su lista de vídeos

Escenario principal:

1. El artista/grupo se dirige a su perfil y se dirige a la pestaña de vídeos
2. El sistema muestra el listado de vídeos actuales y un botón para añadir vídeo
3. El usuario pulsa sobre el botón e introduce nombre y URL del vídeo
4. El sistema almacena los datos en la base de datos

Caso de uso: Modificar vídeo de YouTube
Actor: Artista y grupo
Precondición: El vídeo existe y el artista/grupo es su autor
Disparador: El usuario quiere modificar un vídeo de YouTube de su lista de vídeos
Escenario principal:

1. El artista/grupo se dirige a su perfil y se dirige a la pestaña de vídeos
2. El sistema muestra el listado de vídeos actuales
3. El usuario pulsa sobre el botón de modificar en el vídeo deseado
4. El sistema muestra dos campos: nombre y URL que por defecto muestran los valores actuales
5. El usuario hace los cambios desados y pulsa sobre modificar
6. El sistema almacena los cambios sobre el vídeo en la base de datos

---

Caso de uso: Eliminar vídeo de YouTube
Actor: Artista y grupo
Precondición: El vídeo existe y el artista/grupo es su autor
Disparador: El usuario quiere eliminar un vídeo de YouTube de su lista de vídeos
Escenario principal:

1. El artista/grupo se dirige a su perfil y se dirige a la pestaña de vídeos
2. El sistema muestra el listado de vídeos actuales
3. El usuario pulsa sobre el botón de eliminar en el vídeo deseado
4. El sistema pide confirmación
5. En caso afirmativo, los datos del vídeo se eliminan de la base de datos, en caso contrario todo permanece intacto

---

Caso de uso: Subir foto de perfil
Actor: Fan, artista, grupo y propietario de local
Precondición: El perfil existe
Disparador: El usuario quiere subir una foto para asociarla con su perfil
Escenario principal:

1. El artista/grupo se dirige a su perfil y pulsa sobre la foto actual
2. El sistema muestra opción de subir foto desde galería o de capturar una foto al momento
3. El usuario pulsa la opcion deseada y envía la foto al servidor
4. El sistema guarda la foto en las carpetas del servidor y su nombre en la base de datos

---

Caso de uso: Eliminar foto de perfil
Actor: Fan, artista, grupo y propietario de local
Precondición: El perfil existe y tiene una foto ya asociada a su perfil
Disparador: El usuario quiere eliminar la foto asociada con su perfil
Escenario principal:

1. El artista/grupo se dirige a su perfil y pulsa sobre la foto actual
2. El sistema muestra la opción de eliminar la foto actual además de subir otra foto nueva
3. El usuario pulsa la opcion de eliminar
4. El sistema elimina la foto en las carpetas del servidor y cambia el nombre de la foto de perfil al nombre por defecto

---

Caso de uso: Subir foto de evento
Actor: Propietario de local
Precondición: El evento existe y el perfil de propietario de local es su autor
Disparador: El usuario quiere subir una foto para asociarla con el evento creado
Escenario principal:

1. El usuario se dirige a la página del evento y pulsa sobre la foto actual
2. El sistema muestra opción de subir foto desde galería o de capturar una foto al momento
3. El usuario pulsa la opcion deseada y envía la foto al servidor
4. El sistema guarda la foto en las carpetas del servidor y su nombre en la base de datos

---

Caso de uso: Eliminar foto de evento
Actor: Propietario de local
Precondición: El evento existe y el perfil de propietario de local es su autor
Disparador: El usuario quiere eliminar la foto asociada actualmente al evento
Escenario principal:

1. El usuario se dirige a la página del evento y pulsa sobre la foto actual
2. El sistema muestra la opción de eliminar la foto actual además de subir otra foto nueva
3. El usuario pulsa la opcion de eliminar
4. El sistema elimina la foto en las carpetas del servidor y cambia el nombre de la foto del evento al nombre por defecto

4.6. Iteración 6: Sistema de búsqueda de perfiles y eventos

No se incluye modelo conceptual en esta iteración ya que no se desarrollaron nuevas tablas o relaciones en la base de datos. Si no que tan sólo se diseñaron funciones de búsqueda para perfiles y eventos.

4.6.1. Casos de uso

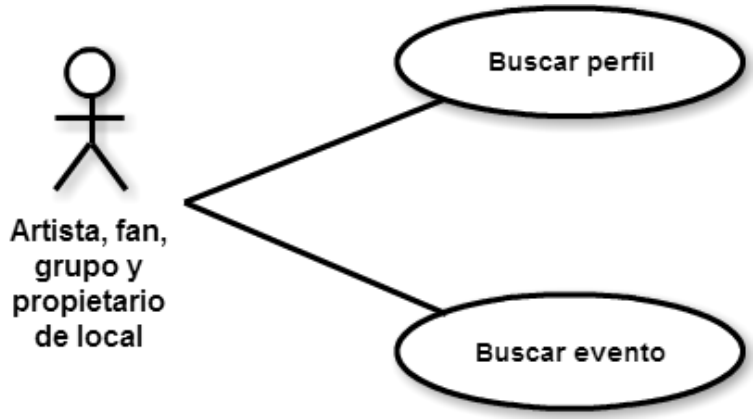

Figura 12: Diagrama de casos de uso de la sexta iteración

Caso de uso: Buscar perfil

Actor: Cualquier perfil

Precondición: Perfil existe

Disparador: El usuario quiere buscar perfiles que encajen en criterios indicados por él

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación y pulsa sobre "buscar perfil"
2. El sistema muestra un formulario
3. El usuario completa el formulario con los datos que él desee y pulsa sobre "buscar"
4. El sistema muestra los perfiles que encajen con la información indicada

---

Caso de uso: Buscar evento

Actor: Cualquier perfil

Precondición: Perfil existe

Disparador: El usuario quiere buscar eventos cercanos a él o que coincidan con información detallada por él

Escenario principal:

1. El usuario despliega el menú lateral de la aplicación y pulsa sobre "eventos"
2. El sistema muestra una pantalla con dos pestañas: "eventos cercanos hoy" y "buscar eventos", en la primera hay un mapa con eventos cercanos, en la segunda se pueden buscar eventos por nombre, provincia, etc.
3. El usuario completa los datos sobre el evento que quiere buscar o bien pulsa sobre el evento en el mapa
4. El sistema muestra la pantalla con información del evento

4.7. Modelo conceptual completo

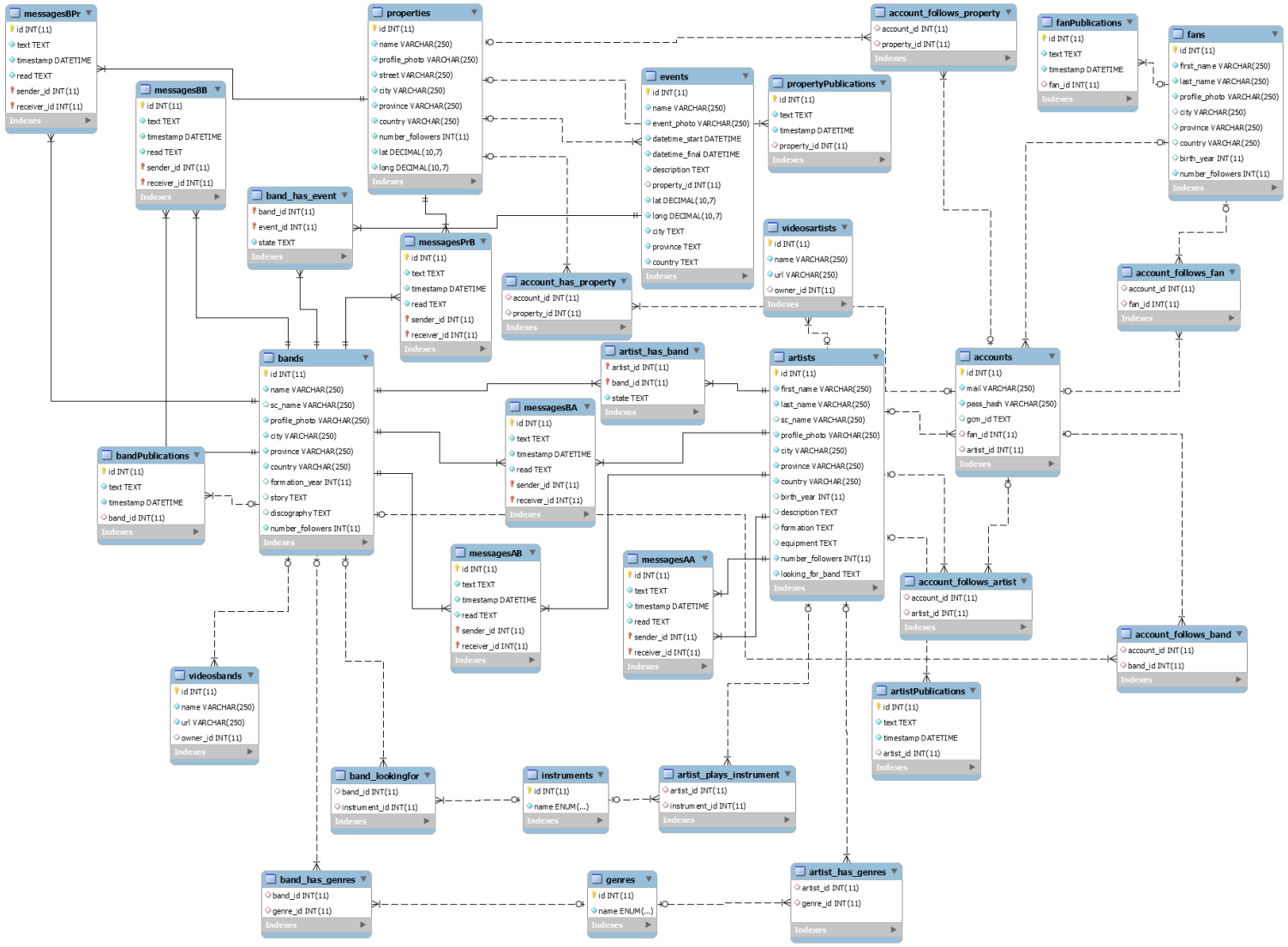

Figura 13: Modelo conceptual completo

Restricciones

- Todas las clases han de tener un identificador
- El campo `timestamp` toma por defecto el instante de tiempo al crearse la instancia
- El campo `read` por defecto es false, este valor cambia cuando se abre el mensaje en la aplicación
- Para una cuenta sólo puede existir un artista o un fan, sólo uno exclusivamente
- El campo `mail` ha de ser único
- Para cada `account` y `property` sólo puede existir una instancia de `account_has_property`
- Para cada `account` y `property` sólo puede existir una instancia de `account_follows_property`
- Para cada `account` y `band` sólo puede existir una instancia de `account_follows_band`
- Para cada `account` y `artist` sólo puede existir una instancia de `account_follows_artist`
- Para cada `account` y `fan` sólo puede existir una instancia de `account_follows_fan`
- Para cada `band` y `artist` sólo puede existir una instancia de `artist_has_band`
- Para cada `band` y `event` sólo puede existir una instancia de `band_has_event`
- Para cada `band` y `genres` sólo puede existir una instancia de `band_has_genres`
- Para cada `artist` y `genres` sólo puede existir una instancia de `artist_has_genres`
- Para cada `band` y `instruments` sólo puede existir una instancia de `band_lookingfor`
- Para cada `artist` y `instruments` sólo puede existir una instancia de `artist_plays_instrument`

5. Diseño

5.1. Tecnología

5.1.1. Servidor

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Flask

Flask es un framework minimalista escrito en Python y basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2. Flask es muy flexible gracias a todos sus *plugins* disponibles que harán escalar, crecer la aplicación y que sea posible añadirle cualquier funcionalidad que deseemos.

MySQL

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

SQLAlchemy [32]

SQLAlchemy es un toolkit de bases de datos SQL para Python y también es un *Object Relational Mapper*, lo que facilita en gran manera la interacción entre la aplicación y la base de datos. Concretamente se ha utilizado el *plugin* Flask-SQLAlchemy [33], el cual añade SQLAlchemy a nuestra aplicación Flask.

5.1.2. Cliente

Android

Android es un sistema operativo basado en Linux. Su uso está diseñado para *smartphones*, *tablets* y también *smartwatches*. Tal y como se puede ver en Kantar World Panel [34] y en la figura 14, tanto en Europa como en América este sistema operativo tiene un uso mayoritario.

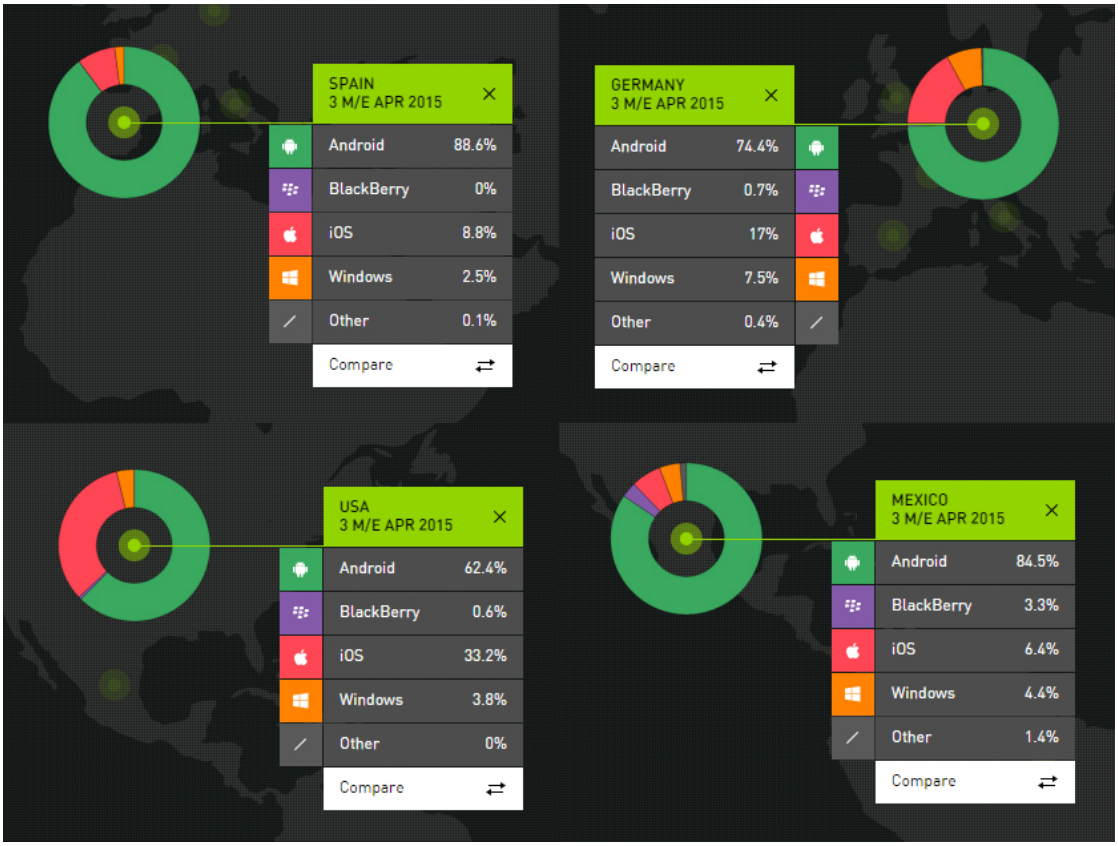

Figura 14: Porcentaje del uso de sistemas operativos en España, Alemania, E.E.U.U y México

AQuery [35]

AQuery es una librería para Android la cual permite hacer de forma más fácil y sencilla tareas asíncronas y manipular elementos de la interfaz de usuario. Aún así, en este

proyecto esta librería se ha utilizado sólo para realizar *requests* a la API del servidor, es decir, no hemos utilizado su capacidad para manipular elementos de la interfaz de usuario.

5.2. Arquitectura

Flask no sigue ningún modelo de arquitectura definido, sin embargo el modelo se define según se añaden *plugins* con lo que se acaba consiguiendo es una arquitectura de Modelo-Vista-Controlador (MVC) [36]. Gracias a esta arquitectura tenemos en capas independientes los datos y su definición, las operaciones que se realizan sobre estos datos tales como lecturas y escrituras y, por último, la vista que se muestra al usuario. Al tener estas capas independientes nos aseguramos de que ningún error en una capa puede afectar a otra.

5.2.1. Modelo

Esta es la capa que representa los datos de la aplicación. Dicha capa ha de ser capaz de permitir a la aplicación realizar operaciones CRUD.

Tal y como se ha dicho anteriormente, se utiliza SQLAlchemy para conectar la aplicación a la base de datos. Gracias a este *toolkit* se consigue:

- Una declaración sencilla de los modelos y las relaciones entre ellos
- Utilizar objetos de la base de datos como si fueran objetos de Python
- Un conjunto de operaciones para utilizar en Python sobre la base de datos

5.2.2. Vista

Esta capa es la encargada de mostrar al usuario la información, además de transferir a la capa de controladores las interacciones entre el usuario y esta capa.

En el caso de una página web, en esta capa tendríamos que el servidor envía al cliente el código HTML para que se visualice en el navegador. En cambio, en este caso tenemos que esta capa es toda la aplicación Android. Esta capa se encarga de recibir los datos en formato JSON desde la capa controlador y los muestra al usuario. La aplicación Android también es encargada de transmitir las interacciones entre el usuario y la aplicación a la capa controlador a través de *requests* HTTP a ciertas rutas que se asocian con operaciones sobre la base de datos.

5.2.3. Controlador

Esta capa contiene todas las operaciones que se realizan sobre los modelos.

En el caso de Flask, estas operaciones están asociadas a rutas. Por ejemplo, la ruta `/bands/1` estaría asociada a la lectura (GET) de la base de datos del grupo de música con el identificador 1. En el caso de una página web esta operación devolvería, por ejemplo, profile.html con los datos referentes a ese grupo, en cambio, al tratarse de una API se devuelve un mensaje en formato JSON con los datos que la capa vista es responsable de extraer y mostrar.

6. Implementación

A la hora de desarrollar el sistema, primero se desarrolló la parte del servidor y posteriormente la parte Android. Es decir, primero se desarrollaron las capas de modelo y de controlador.

Por cada una de las 6 iteraciones de desarrollo del servidor se implementaron las 2 capas anteriormente nombradas. De esta forma se declaraban los modelos necesarios para esa iteración y también las operaciones necesarias para crear, leer y modificar dichos datos en los modelos.

6.1. Servidor

Con el objetivo de aclarar al máximo como ha sido el desarrollo del servidor se mostrará los trozos de código más destacables y variados.

6.1.1. Modelo

En el código 1 tenemos el ejemplo de como declarar el modelo `Account`. Por cada campo del modelo especificamos si es una columna (`db.Column`) o bien una relación con otro modelo (`db.relationship`). También se pueden especificar opciones como si la columna es *primary key* como se puede ver en la línea 8, si el campo puede ser nulo o no, si es único y si es índice de la tabla, es decir, si la tabla se ha de ordenar respecto a ese campo (todas estas opciones se pueden ver en la línea 9).

Para crear una relación `one-to-one` entre dos tablas tan solo hemos de indicar:

- En una tabla, especificar que la columna es una *foreign key*, para ello se especifica de la forma que se puede ver en las líneas 12 y 13, indicando el nombre de la tabla (campo tablename de dicha tabla) seguido por el campo de esa tabla, en este caso el identificador

- En la otra tabla, indicamos el campo como *relationship* (línea 29) e indicamos 3 cosas: el nombre de la tabla, el nombre de la referencia inversa (en caso que desde un objeto `Account` queramos acceder a un objeto `Fan`) y el parámetro `uselist` como falso

```
1  account_has_property = db.Table('account_has_property',
2      db.Column('account_id', db.Integer, db.ForeignKey('accounts.id')),
3      db.Column('property_id', db.Integer, db.ForeignKey('properties.id'))
4  )
5
6  class Account(db.Model):
7      __tablename__ = 'accounts'
8      id = db.Column(db.Integer, primary_key=True)
9      mail = db.Column(db.String(250), nullable=False, index=True, unique=True)
10     pass_hash = db.Column(db.String(250), nullable=False)
11     gcm_id = db.Column(db.Text)
12     fan_id = db.Column(db.Integer, db.ForeignKey('fans.id'))
13     artist_id = db.Column(db.Integer, db.ForeignKey('artists.id'))
14     properties = db.relationship('Property',
15                                  secondary=account_has_property,
16                                  backref=db.backref('owner', lazy='dynamic'),
17                                  lazy='dynamic')
18     following_fans = db.relationship('Fan',
19                                  secondary=account_follows_fan,
20                                  backref=db.backref('fan_followed_by', lazy='dynamic'),
21                                  lazy='dynamic')
22     ...
23
24 class Fan(db.Model):
25     __tablename__ = 'fans'
26     id = db.Column(db.Integer, primary_key=True)
27     first_name = db.Column(db.String(250), nullable=False, index=True)
28     ...
29     account = db.relationship('Account', backref='fan', uselist=False)
30     ...
```

Código 1: Código de ejemplo de como definir un modelo y sus relaciones en Flask-SQLAlchemy

Respecto a la siguiente relación, la `one-to-many` se realiza de la misma forma pero con el campo `uselist` a cierto. El valor por defecto de dicho campo por defecto es cierto, así que no hace falta especificar nada en este caso.

En lo que a último tipo de relación corresponde, si se siquiera crear una relación `many-to-many` es necesario crear una tabla asociativa y luego especificar una relación a dicha tabla. En este caso podemos ver la tabla asociativa `account_has_property` en la que se incluyen dos columnas, las cuales son *foreign keys* a las tablas que asocian, en este caso `account` y `property`. El siguiente paso es crear en una tabla o en la otra una relación a esta tabla asociativa. Dicha relación se puede ver en la línea 14 y 18 (código 1). En esta relación se especifica: el nombre de la clase a la que hace relación la columna, la tabla asociativa que hemos creado, y el nombre de la referencia inversa para que desde la otra tabla (en la cual no declaramos ninguna relación) se pueda acceder al otro objeto.

En ocasiones, cuando se tiene una relación `many-to-many`, también se quiere asociar un dato a dicha relación. Dicho dato se puede tratar de, por ejemplo, un mensaje. Un mensaje no deja de ser una relación entre dos usuarios adjuntada con un texto y un `timestamp`. En este caso, hay que crear el modelo mensaje como una clase y crear dos relaciones `one-to-many` tal y como se ha explicado anteriormente.

Por último, dentro de estas clases que representan los modelos se han declarado unas funciones que sirven para exportar e importar los datos de dicho modelo, como podemos ver en el código 2. En la línea 4 tenemos la función de exportar, que devuelve un `Dictionary` de Python. Por otro lado, para importar los datos tenemos la función `import_data` en la línea 19. Esta función, además de tener como parámetros el propio objeto también tiene los datos, que generalmente son los datos recibidos en el *request*. Se comprueba si dicho campo existe en los datos recibidos, y si lo está, sobreescribe el valor actual por ese. La funcion `format_string` es una función definida que sirve para eliminar espacios a final y a inicio de palabra en caso de que el usuario los haya introducido sin querer y pone la primera letra como mayúscula si no lo está.

```python
class Fan(db.Model):
    ...

    def export_data(self):
        return {
            'id': self.id,
            'account_id': self.account.id,
            'self_url': self.get_url(),
            'first_name': self.first_name,
            'last_name': self.last_name,
            'profile_photo': self.profile_photo,
            'city': self.city,
            'province': self.province,
            'country': self.country,
            'number_followers': self.number_followers,
            'birth_year': self.birth_year
        }

    def import_data(self, data):
        try:
            if 'first_name' in data:
                self.first_name = format_string(data['first_name'])
            if 'last_name' in data:
                self.last_name = format_string(data['last_name'])
            if 'city' in data:
                self.city = format_string(data['city'])
            if 'province' in data:
                self.province = format_string(data['province'])
            if 'country' in data:
                self.country = format_string(data['country'])
            if 'birth_year' in data:
                self.birth_year = data['birth_year']
        except KeyError as e:
            raise ValidationError('Invalid account: missing ' + e.args[0])
        return self
```

Código 2: Código de las funciones export_data e import_data

6.1.2. Controlador

Una vez creados los modelos, sus relaciones entre ellos y estos métodos auxiliares que ayudarán a expoertar e importar los datos de dichos modelos, se ha de empezar a desarrollar todas las operaciones sobre dichas tablas. Para ello se asocia una función Python

a una ruta en la API y un método HTTP.

En el código 3 tenemos el ejemplo más simple de operaciones CRUD sobre la tabla de fans. Antes de nada, recordemos que el método HTTP GET es para lectura, POST para crear, PUT para modificar y DELETE para eliminar. Como se puede ver, cada función está asociada a una ruta de la API y además se pueden especificar variables y también a un método.

Primero, para la función `get_fan` se ejecuta la `query get_or_404`, este método viene definido en *Flask-SQLAlchemy* y lo que hace es buscar en dicha clase la ID especificada y, en caso de no existir devolver un error 404. Al realizar este *query* se obtiene un objeto de la clase `Fan`, por lo que si se llama al método `export_data` se obtendrá el diccionario de ese fan. Por otro lado, `Jsonify` es un módulo de *Flask*, es similar a la función de Python `json.dumps` pero en vez de devolver un `string` como este último devuelve un objeto `flask.response` con el header correspondiente `Content-Type: application/json`.

En la segunda función se puede ver como crear una instancia de Fan asociada a una cuenta existente. Lo primero que se hace es buscar en la base de datos la cuenta especificada como variable en la ruta gracias a la función `get_or_404`. Posteriormente se crea una nueva instancia de Fan y se le asigna al campo account (el cual es un `db.relationship`) la cuenta que se ha encontrado en la base de datos. seguidamente se hace un `import_data` pasando como datos todo el json del `request`, se añade el nuevo objeto fan a la base de datos y se realiza un `commit`. Como respuesta a la petición se devuelve un `body` vacío y en las cabeceras la localización del nuevo fan tal como su identificador.

Para los dos últimos métodos, `edit_fan` y `delete_fan`, primero se consigue la instancia de fan deseada con el `query get_or_404` y seguidamente se importan los datos del request y se añade a la base de datos en el caso de modificación o se borra de la base de datos en el caso del `delete`.

```python
@api.route('/api/fans/<int:id>', methods=['GET'])
def get_fan(id):
    return jsonify(Fan.query.get_or_404(id).export_data())

@api.route('/api/accounts/<int:id>/fan', methods=['POST'])
def new_account_fan(id):
    account = Account.query.get_or_404(id)
    fan = Fan(account=account)
    fan.import_data(request.json)
    db.session.add(fan)
    db.session.commit()
    return jsonify({}), 201, {'Location': fan.get_url(), 'id': fan.id}

@api.route('/api/fans/<int:id>', methods=['PUT'])
def edit_fan(id):
    fan = Fan.query.get_or_404(id)
    fan.import_data(request.json)
    db.session.add(fan)
    db.session.commit()
    return jsonify({})

@api.route('/api/fans/<int:id>', methods=['DELETE'])
def delete_fan(id):
    fan = Fan.query.get_or_404(id)
    db.session.delete(fan)
    db.session.commit()
    return jsonify({})
```

Código 3: Código de ejemplo de operaciones CRUD sobre un modelo

Después de haber visto los ejemplos más sencillos de operaciones, ahora vamos a ver dos ejemplos de búsqueda de un perfil. En el código 4 se pueden ver las funciones para buscar un fan y un artista.

Para buscar un fan sólamente se ha de utilizar la función `filter_by` de *Flask-SQLAlchemy*. En esta función normalmente se ha de especificar algo como `filter_by(name='toni')` pero en este caso los parámetros que recibimos del request ya tienen ese formato, asi que se puede pasar dicho objeto `request.args` a la función. Finalmente, solo se ha de devolver dicha lista a través de jsonify.

Para buscar un artista y/o grupo no podemos utilizar la función `filter_by` sin más ya que los parámetros `genre1`, `genre2`, `genre3`, `instrument1`, `instrument2` e `instrument3` no existen en la clase `Artist`. Así que primero separamos estos campos en distintos diccionarios, buscamos una lista de artistas que coincida con los otros campos y después se comprueba si esos artistas tocan esos instrumentos y/o géneros.

```python
@api.route('/api/fans/search', methods=['GET'])
def search_fan():
    result = Fan.query.filter_by(**request.args).all()
    return jsonify({'fans': [fan.export_data() for fan in result]})

@api.route('/api/artists/search', methods=['GET'])
def search_artist():
    dict = request.args.copy()
    params = MultiDict([])
    genres = MultiDict([])
    instruments = MultiDict([])
    for key in dict:
        if key in ['first_name', 'last_name', 'city', 'province', 'country',
            'looking_for_band']:
            params.add(key, dict.get(key))
        elif key in ['genre1', 'genre2', 'genre3']:
            genres.add(key, dict.get(key))
        elif key in ['instrument1', 'instrument2', 'instrument3']:
            instruments.add(key, dict.get(key))

    result = Artist.query.filter_by(**params).all()

    for artist in result:
        genrenames = []
        for genre in artist.genres:
            genrenames += [genre.name]
        for name in genres.values():
            if name not in genrenames:
                result.remove(artist)

    for artist in result:
        instrumentnames = []
        for instrument in artist.instruments:
            instrumentnames += [instrument.name]
        for name in instruments.values():
            if name not in instrumentnames:
                result.remove(artist)

    return jsonify({'artists': [artist.export_data() for artist in result]})
```

Código 4: Código de ejemplo de operaciones de búsqueda sobre perfiles, fan y artista en este caso

6.2. Cliente

La aplicación Android forma, en su totalidad, toda la capa vista. Los elementos gráficos que forman una interfaz gráfica en Android son *widgets*, campos de texto o contenedores distruibidos en un *layout*. Para comunicar la aplicación cliente con el servidor se utiliza AQuery.

6.2.1. Vista

Primero, en el código 5 se muestra una petición realizada a través de AQuery. Gracias a esta librería se puede realizar tanto peticiones GET, POST, PUT como DELETE. En concreto, en el código 5 se realiza una petición GET. Si se quisiera realizar una petición PUT o POST, se ha de cambiar el método `ajax` por `post` o `put` y añadir un `JSONObject` como parámetro después de la url y en caso de querer realizar un DELETE, utilizar el método `ajax` y utilizar `.method(AQuery.METHOD_DELETE)` sobre AjaxCallback. Por último, destacar el uso del método `auth`, este método se utiliza para añadir cabeceras de autentificación y se usa para adjuntar correo electrónico y contraseña en la pantalla de login y, posteriormente, el token en el resto de llamadas a la API.

```
AQuery aq = new AQuery(this);

BasicHandle handle = new BasicHandle(user, password);

aq.auth(handle).ajax(url, JSONObject.class, new AjaxCallback<JSONObject>() {
    @Override
    public void callback(String url, JSONObject object, AjaxStatus status) {
        if (object != null) {
            // Response recibido -> object
        }
        else {
            if (status.getCode() == 500) {
                // Mostrar error 500
            }
            else if (status.getCode() == 401) {
                // Mostrar error 401
            }
            else if (status.getCode() == 404) {
                // Mostrar error 404
            }
            else {
                // Mostrar error inesperado
            }
        }
    }
});
```

Código 5: Código de ejemplo una petición GET con AQuery

Después de haber visto el método que utilizaremos para comunicarnos con el servidor, se mostrarán las pantallas de la aplicación tal y como código destacable de cada una de ellas.

La primera pantalla que aparece al iniciar la aplicación es la de login. En esta, tenemos dos opciones: pulsar sobre “Regístrate” para llevarnos a la pantalla de registro o rellenar los campos pedidos y pulsar sobre “Login” para ingresar en la aplicación. En la pantalla de registro se comprueba que las contraseñas coincidan y que el usuario sólo se pueda registrar en caso de aceptar la política de privacidad, en caso contrario aparecerá un mensaje informativo reportando el error. Para enviar los datos de login se utiliza el

objeto `BasicHandle` tal y como se ha dicho en el apartado anterior. En cambio, para enviar los datos de registro al servidor se crea un `JSONObject` que se pasa como parámetro con AQuery. A la hora del registro de un usuario también se registra su dispositivo en el servidor de *Google Cloud Messaging* para poder enviarle notificaciones, así que además de enviar al servidor su usuario y contraseña de registro también se envía el identificador del dispositivo una vez registrado.

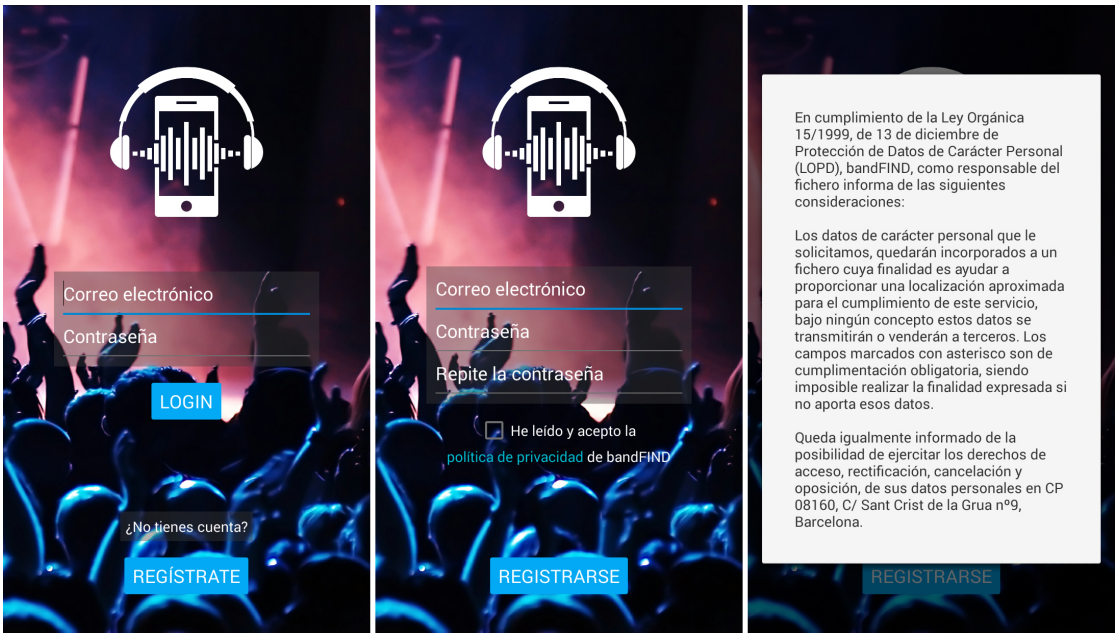

Figura 15: Pantallas de login, registro y política de privacidad

Cuando el usuario se loguea pueden pasar 3 cosas: que el usuario no tenga ningún perfil creado, que tenga un perfil creado o que tenga más de uno. Estos 3 casos se detectan y se actúa al respecto. En el primer caso se muestra una pantalla explicando lo sucedido y pidiendo al usuario que cree un perfil. Como se puede ver, en la parte superior de la pantalla hay un menú desplegable en el cual se podrá elegir uno entre los perfiles posibles a crear y aparacerá un formulario que el usuario tendrá que rellenar (al menos los campos obligatorios). En el segundo caso se entrará a la pantalla principal de la aplicación con ese único perfil y, en el tercer caso se mostrará una pantalla con sus

perfiles y podrá elegir con cual ingresar.

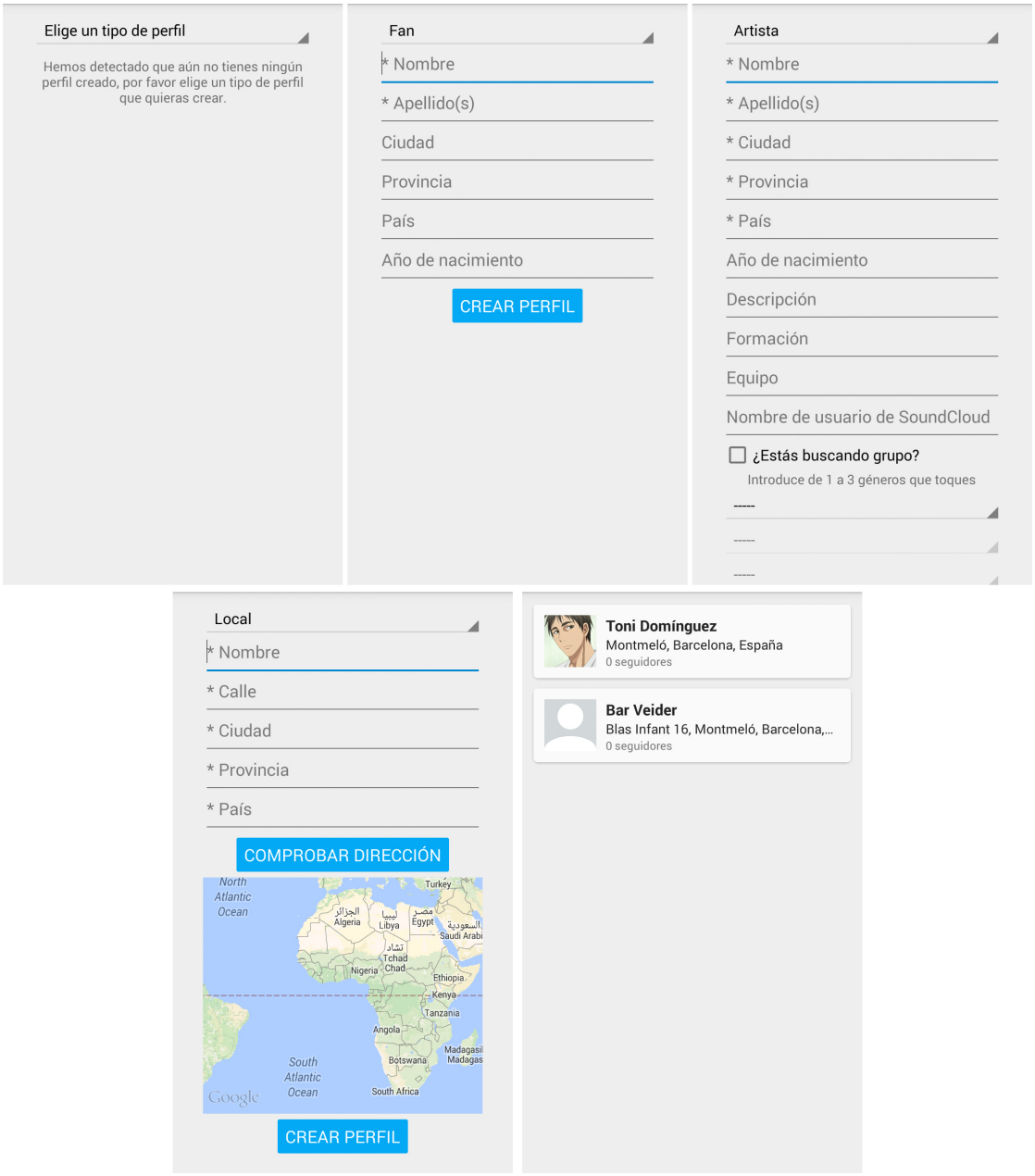

Figura 16: Pantallas de creación de perfil y pantalla de elección de perfil al entrar en la aplicación

Inicio

Una vez hayamos ingresado con un perfil, accederemos a la pantalla principal. Esta contiene varias funcionalidades: campo de texto para publicar noticias, *feed* de publicaciones de los perfiles que el usuario sigue, además de sus propias publicaciones, por lo que cuando publique algo lo verá al instante y también podrá modificar o eliminar sus publicaciones e invitaciones recibidas tanto de eventos como de grupos. Por último, además de todas estas funcionalidades en esta pantalla, podemos desplegar el menú lateral el cual permite al usuario navegar hasta otras pantallas.

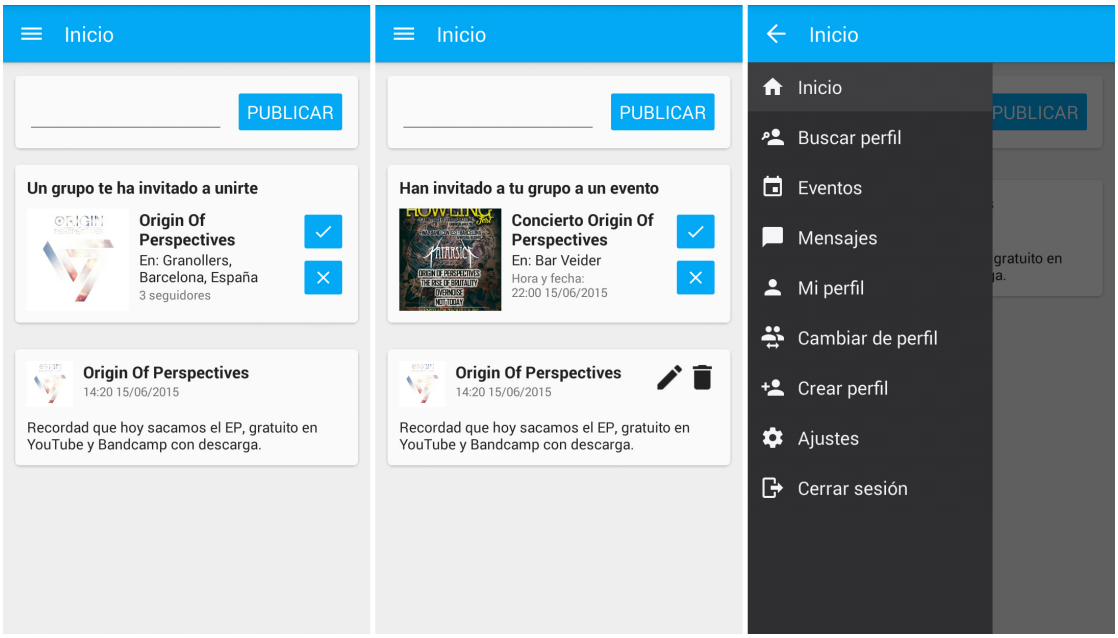

Figura 17: Pantalla de inicio: invitación de un grupo recibida, invitación de un evento recibida, menú desplegable

Buscar perfil

La funcionalidad de buscar perfil se puede ejecutar desde cualquier tipo de perfil. Al elegirla desde el menú se abre una pantalla que permite al usuario elegir que tipo de perfil quiere buscar y un formulario para, al menos, completar un campo que limite la búsqueda. Al pulsar el botón de buscar aparece otra pantalla con la lista de resultados que encajan mostrando información básica sobre el perfil y con la capacidad de visitar el perfil si se pulsa sobre él. Por último, destacar que el propio perfil que esté realizando

la búsqueda, como es obvio, no estará incluido en los resultados en caso de coincidir en los parámetros de búsqueda.

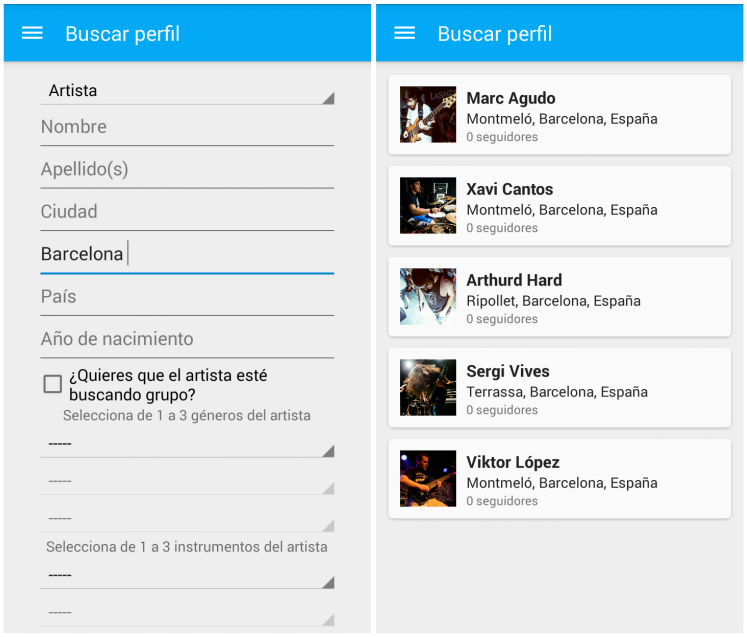

Figura 18: Pantalla para buscar un perfil, en este caso de artista, y resultados de la búsqueda

En cuanto al código de esta funcionalidad, para realizar dicha búsqueda se realiza una petición GET con los parámetros en la URL. Esto se consigue de la siguiente forma:

```
List<NameValuePair> params = new LinkedList<NameValuePair>();

params.add(new BasicNameValuePair("first_name", name);
params.add(new BasicNameValuePair("last_name", surname);
params.add(new BasicNameValuePair("city", city);
params.add(new BasicNameValuePair("province", province);
params.add(new BasicNameValuePair("country", country);
params.add(new BasicNameValuePair("birth_year", bYear);

String paramString = URLEncodedUtils.format(params, "utf-8");
String urlsearch = pref.getString("serverip", null) + "/api/fans/search?" + paramString;

aq.auth(handle).ajax(urlsearch, JSONObject.class, new AjaxCallback<JSONObject>(){ ... });
```

Código 6: Código para añadir párametros a una URL para una petición GET

Posteriormente, el objeto JSON recibido se parsea a un JSONArray. Dichos objetos del JSONArray se pasan a un ArrayList de Java y, gracias a una clase ArrayAdapter se mapean a un ListView de Android.

Eventos

Al pulsar sobre la opción de eventos se abre una pantalla con dos pestañas: Eventos cercanos hoy y Agenda de eventos.

La primera pestaña muestra un mapa que, por defecto, se centrará en la ubicación GPS del usuario y mostrará con unos marcadores los eventos que tengan lugar el mismo día y en un radio de 5 kilómetros. Si pulsa sobre el diálogo del evento se accede a la página de información del evento.

La segunda pestaña abre un formulario para que el usuario pueda buscar por nombre o ubicación. Al pulsar en buscar aparecerá un diccionario, en este aparecerán marcados los días en los que hay al menos un evento y cuando el usuario pulse sobre ese día aparecerá en la parte inferior la lista de evento(s) realizado(s) ese día. Al pulsar sobre cualquier evento se accede a la página de información del evento.

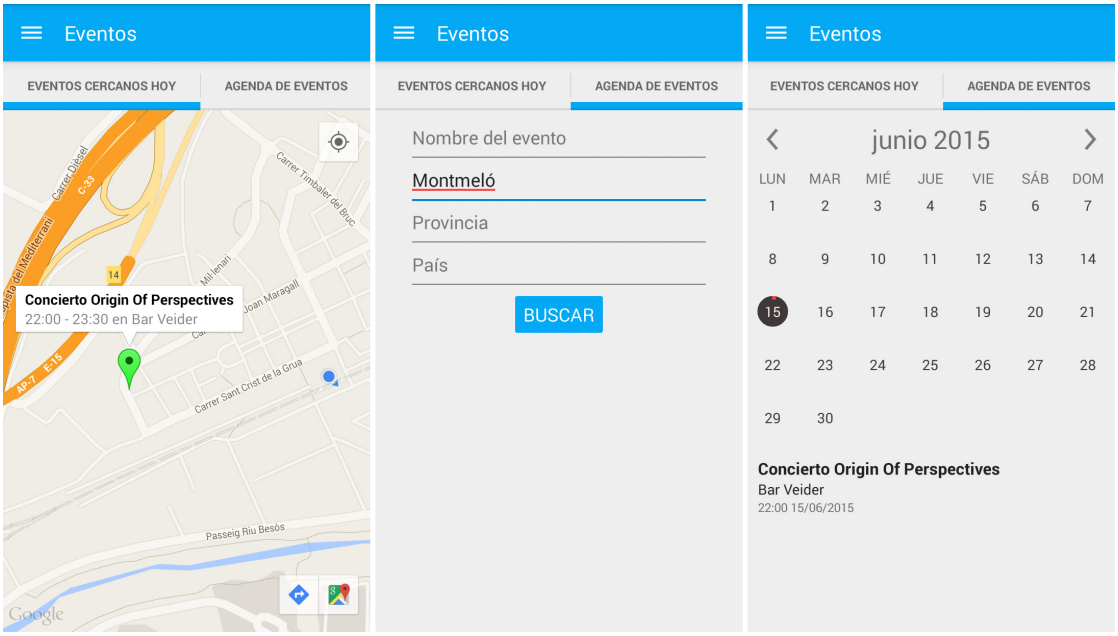

Figura 19: Mapa mostrando eventos cercanos hoy, pantalla de buscar evento y resultado de búsqueda

Desde la pantalla de eventos se muestra información del mismo tal y como el local en el que se celebra, la descripción del mismo y los grupos participantes.

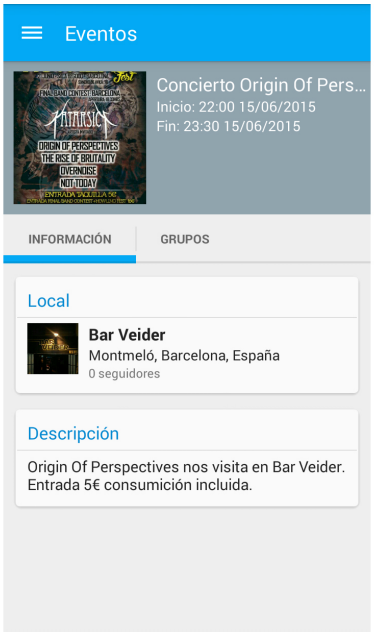

Figura 20: Pantalla de información de un evento

Mensajes

La funcionalidad de mensajes no es accesible desde un perfil de fan. El propósito de la aplicación es que se comuniquen artistas, grupos y propietarios de locales. Cuando desde uno de estos perfiles se pulse sobre mensajes se abre una pantalla listando todos los mensajes recibidos, ordenados desde el más reciente al más antiguo y clasificados en el tipo de perfil del usuario que ha enviado el mensaje. Al pulsar sobre un mensaje en concreto aparecerá el texto al completo y un botón para responder al usuario.

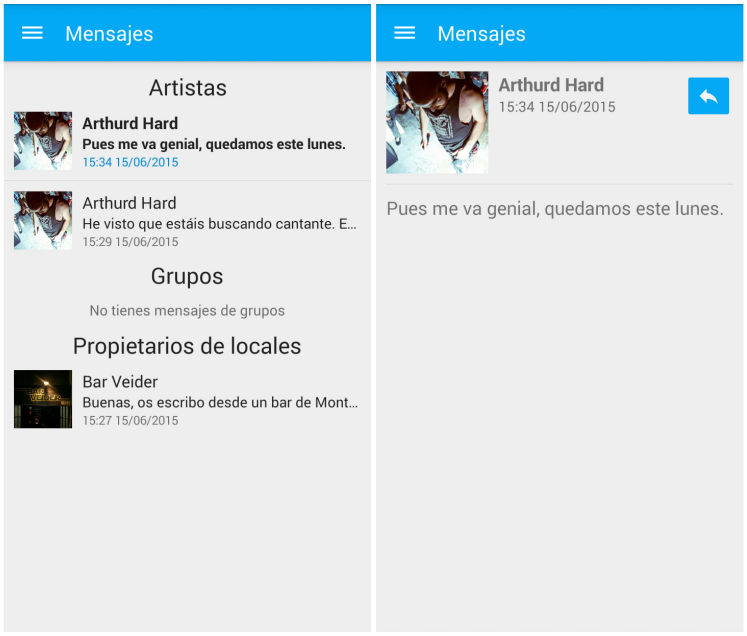

Figura 21: Pantallas de mensajes recibidos y de visualizar mensaje

Mi perfil

La página de perfil es una de las pantallas más importantes de la aplicación. Gracias a esta pantalla se puede conocer toda la información refente a un usuario además de una serie de funcionalidades extras dependiendo el tipo de perfil. También es desde el perfil donde se puede cambiar la foto de perfil que nos representa.

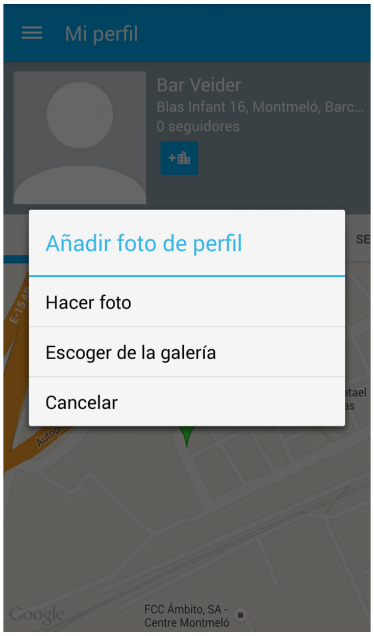

Figura 22: Diálogo para escoger foto de perfil

Un perfil de fan mostrará sus publicaciones, sus seguidores y sus seguimientos.

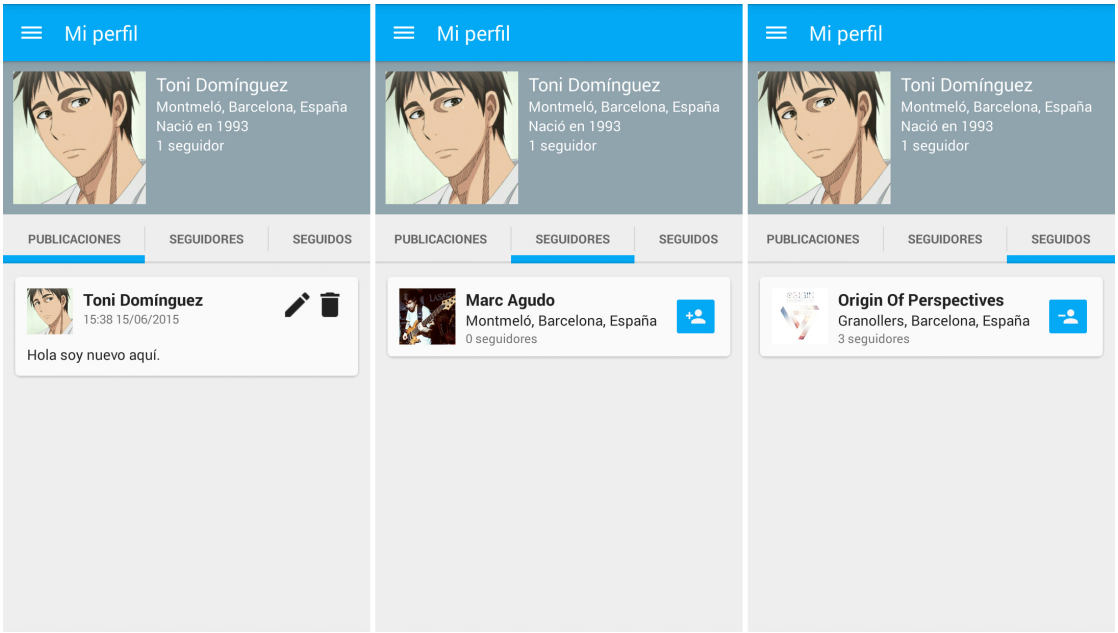

Figura 23: Pantalla de perfil de fan

Un perfil de artista mostrará su información tal y como perfil de SoundCloud, descripcion, formación, géneros, instrumentos, etc. También mostrará los grupos en los cuales está, sus publicaciones, sus vídeos de YouTube además de poder permitir al propio artista la capacidad de subirlos, sus seguidores y sus seguimientos.

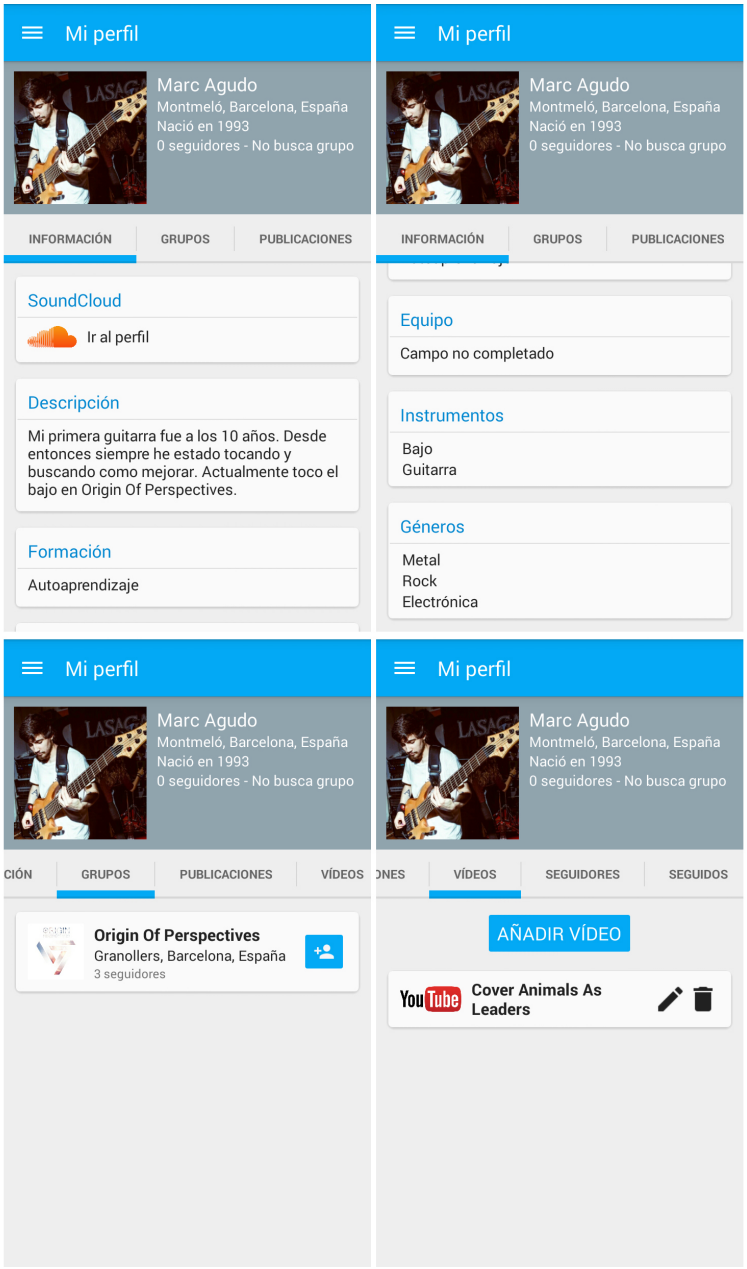

Figura 24: Pantalla de perfil de artista

Un perfil de grupo es similar al de artista, tan sólo que se cambia la función de ver grupos del artista por ver los miembros de un grupo y se pueden visualizar también los eventos en los que participa.

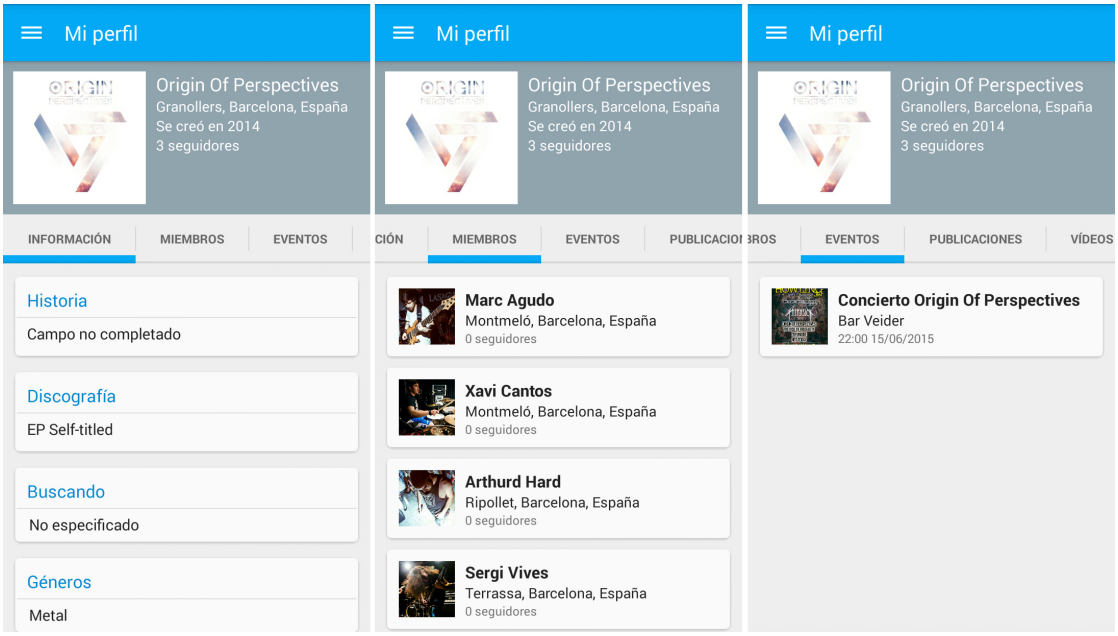

Figura 25: Pantalla de perfil de grupo

Por último, el perfil de propietario de local muestra un mapa de su localización, eventos que tendrán lugar en dicho local y la capacidad de crearlos desde el propio perfil, también sus publicaciones y seguidores.

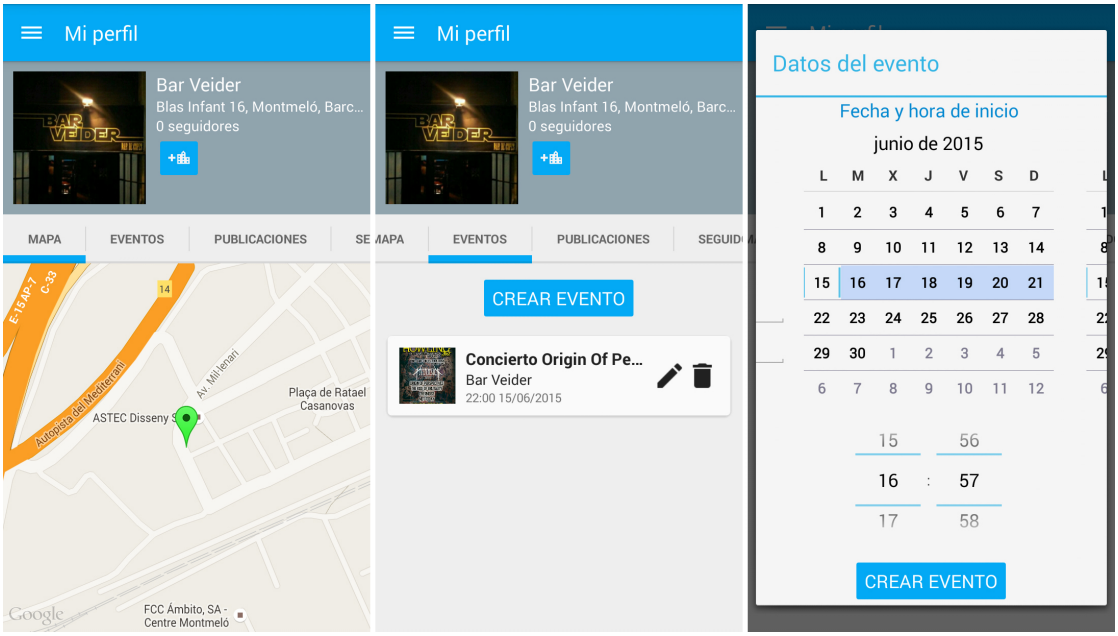

Figura 26: Pantalla de perfil de local

Además de toda la información que se muestra en los perfiles, estos también sirven para interactuar entre ellos: botón de seguir/dejar de seguir, botón de enviar mensaje, botón de añadir propietario a local, botón de añadir artista a grupo y de añadir grupo a evento.

Cambiar de perfil

Esta funcionalidad hace lo que su nombre indica. Al pulsar sobre dicha opción aparece una pantalla que nos permitirá cambiar entre perfiles de la misma cuenta.

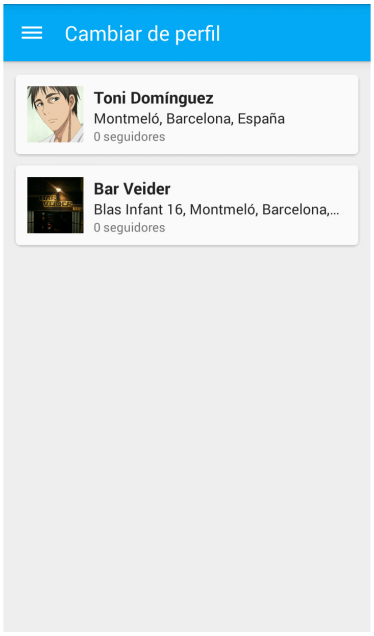

Figura 27: Pantalla para cambiar entre perfiles de una cuenta

Crear perfil

Esta opción permite a cualquier usuario crear un perfil adicional. Desde aquí, accediendo desde un perfil de tipo artista, es la única forma de crear un grupo. Además también se permite el cambio de perfil de artista a fan y viceversa. Este cambio conserva tanto los seguidores como los seguimientos.

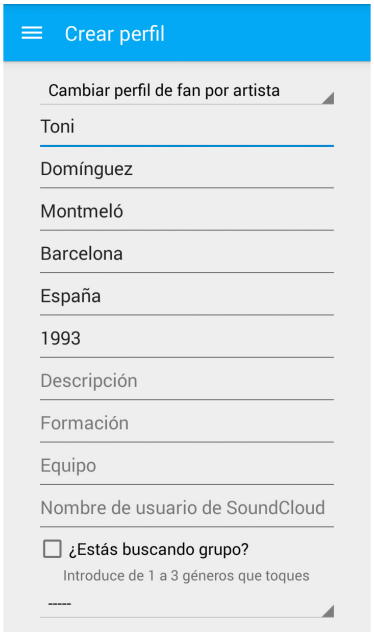

Figura 28: Pantalla de creación de perfiles, en este caso para cambiar de artista a fan

Ajustes

Desde la pantalla de ajustes podemos acceder a las opciones de modificar cuenta o perfil y eliminar cuenta o perfil. Las opciones de eliminar cuenta o perfil cuentan con una confirmación del usuario en caso de que este haya pulsado sin querer. Además, las opciones de modificar cuenta o perfil se rellenan automáticamente con los campos actuales.

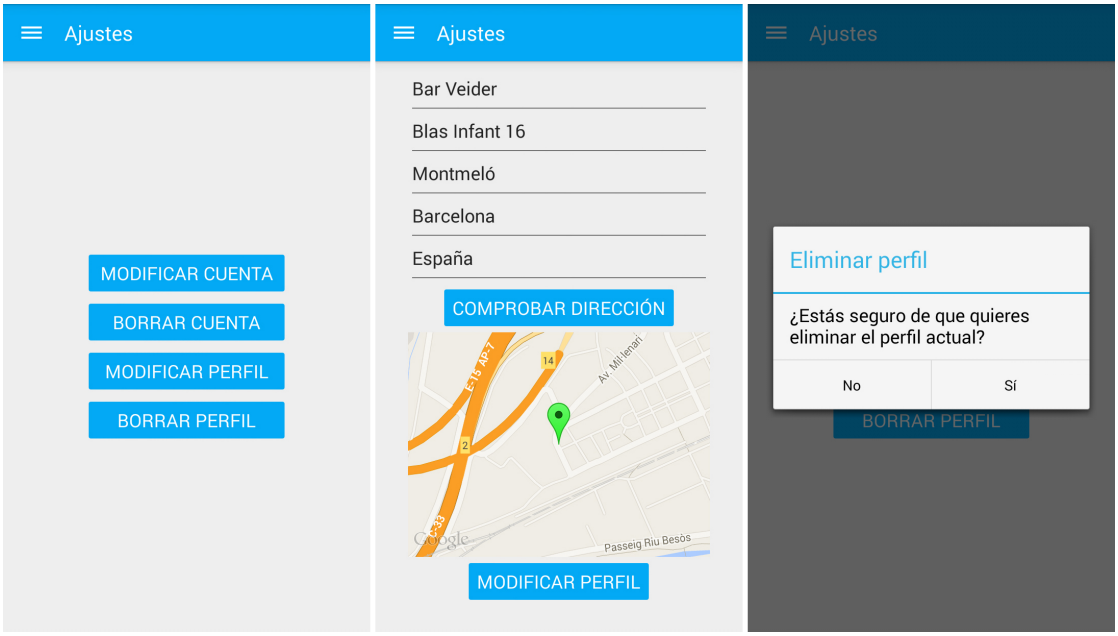

Figura 29: Pantallas de ajustes, modificar perfil y dialogo de confirmación para eliminar perfil

Cerrar sesión

Por último, tenemos la opción de cerrar sesión. Esta opción elimina todos los datos referentes a la sesión del usuario actual, cierra la actividad Android actual y vuelve a abrir la pantalla de login.

6.3. APIs externas

Junto con el desarrollo del proyecto se han utilizado dos APIs externas, concretamente de Google: *Google Cloud Messaging for Android* y *Google Maps Android API v2*. La primera se utiliza con el objetivo de que el usuario reciba notificaciones aún cuando tenga la aplicación cerrada. La segunda se utiliza para poder mostrar mapas en la aplicación junto con marcadores.

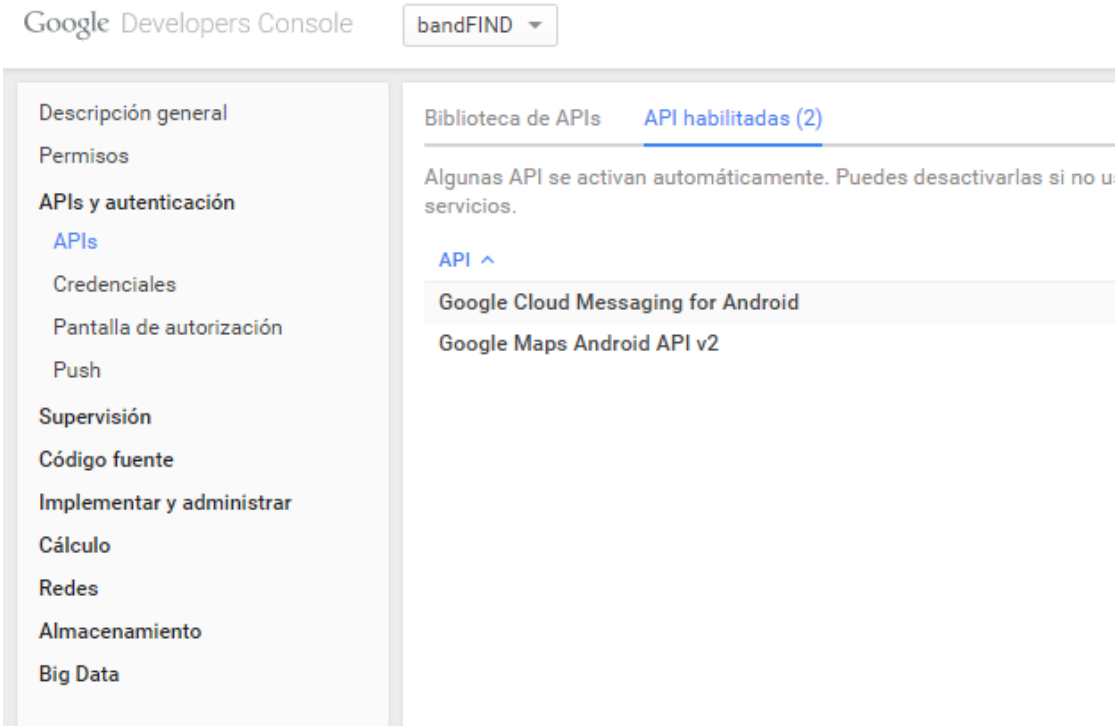

Figura 30: Pantalla que muestra las APIs habilitadas para el proyecto creado

Para poder usar ambas APIs primero hay que crear un proyecto en la *Google Developers Console* y posteriormente activar ambas APIs en la sección Biblioteca de APIs desde APIs y autentificación. Una vez habilitadas ambas APIs hay que crear unas claves y asociarlas al dispositivo Android en caso de la *Google Maps Android API v2* y al servidor en el caso de *Google Cloud Messaging for Android*.

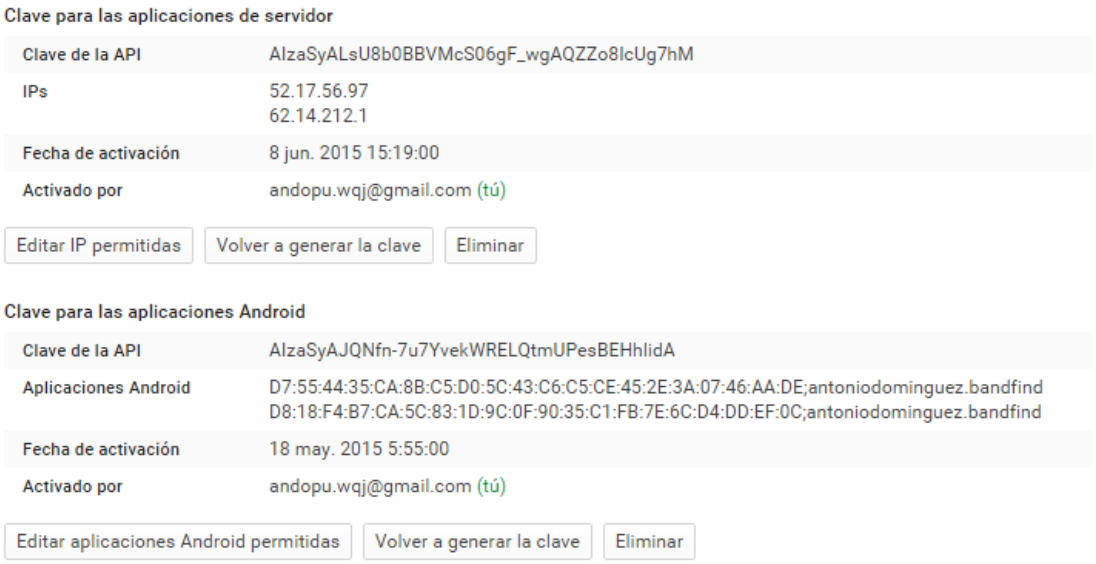

Figura 31: Pantalla que muestra las claves para los servidores y aplicaciones Android especificadas

Para poder asociar un servidor a la clave de la API tan sólo hay que introducir las IPs públicas de los servidores que harán peticiones al servidor *Google Cloud Messaging*. Para asociar una aplicación Android que hace peticiones a la *Google Maps Android API v2* el sistema es un poco más complicado. Hay que especificar la firma SHA1 de la aplicación además del nombre del *package* de la aplicación. Tal y como se puede ver en la figura 31 tenemos dos servidores asociadas (Amazon EC2 [37] y el servidor local) y dos aplicaciones. Las dos aplicaciones son: una con la firma del apk-debug y la otra con la firma del apk-release. Para conseguir ambas firmas hay que utilizar la herramienta `keytool` con los comandos del código 7 (apk-debug y apk-release correspondientemente).

```
keytool -exportcert -alias androiddebugkey
    -keystore C:\Users\Toni\.android\debug.keystore -list -v
keytool -exportcert -alias "alias creado en la keystore cuando
    se ha realizado el apk-release" -keystore "ruta a la keystore creada" -list -v
```

Código 7: Comandos para conseguir la firma SHA1 de apk-debug y apk-release

Dicho comando nos genera una salida con varias firmas, copiamos la firma SHA1 y ya

se puede introducir junto con el nombre del package (separados por ;) para dar acceso a esa aplicación a la API.

6.3.1. Google Cloud Messaging for Android

El funcionamiento de GCM es tal y como se explica en la figura 32.

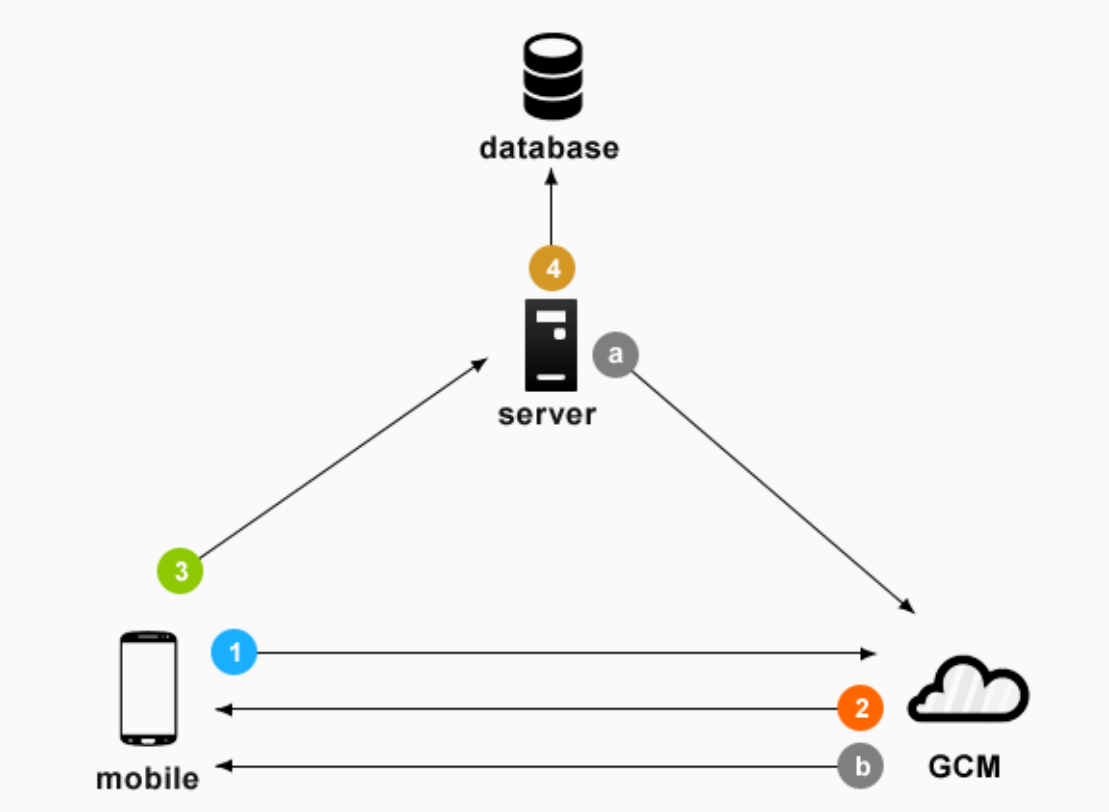

Figura 32: Diagrama del uso de Google Cloud Messaging

1. El disposivo android envía el *sender* id y el *application id* al servidor GCM para poder registarse
2. Después de un exitoso registro el servidor devuelve un *registration id*
3. Después de recibir el *registration id* el dispositivo lo envía al servidor

4. El servidor guarda el *registration id* en la base de datos para un uso posterior

   a) Cuando una se necesita una notificación *push* el servidor envía un mensaje al servidor GCM junto con el *registration id* del dispositivo

   b) El servidor GCM entrega el mensaje al dispositivo correspondiente con ese *registration id*

En el caso de este proyecto, cada vez que un usuario se registra también lo hace su dispositivo de forma transparente al usuario. De esta forma, en la base de datos se tiene cada cuenta asociada con un *registration id*. Cuando un usuario envía un mensaje a otro, se mira si el receptor es un grupo, un local, un artista o un fan. En el caso del grupo o local se envía al servidor GCM un mensaje con el *registration id* de todos los componentes o de los propietarios del local. Por otro lado, si se trata de un artista o fan sólamente se envía a un *registation id*. Lo mismo ocurre cada vez que un grupo invitta a un artista a entrar o cuando un propietario de local invita a un grupo a un evento.

6.3.2. Google Maps Android API v2

Para usar esta API es suficiente con declarar un mapa en un *layout* como en el código 8.

```
<fragment android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.SupportMapFragment" />
```

Código 8: Código para definir un mapa en una pantalla de la aplicación Android

Después sólo hay que cargar el mapa en la clase que se utilice y usar las funciones de esta API para manipularlo. La función del código 9, por ejemplo, centra el mapa en la posición *latitude, longitude*, hace zoom y coloca un marcador verde en dicha posición.

```
protected void loadMap(GoogleMap googleMap) {
    if (googleMap != null) {
        LatLng latLng = new LatLng(latitude, longitude);
        CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(latLng, 16);
        googleMap.animateCamera(cameraUpdate);
        BitmapDescriptor defaultMarker =
            BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN);
        Marker mapMarker = googleMap.addMarker(new MarkerOptions()
            .position(latLng)
            .title(name)
            .icon(defaultMarker));
    }
}
```

Código 9: Código de ejemplo de como manipular el mapa de la API de Google

6.4. Despliegue

Para el despliegue del servidor y poner dicho servicio en producción se ha utilizado Apache 2 en una máquina virtual de Amazon EC2. Antes de nada, para realizar este paso hay que realizar un cambio en una línea de código del servidor. En *Flask* existe una línea de código que es la que ejecuta el servidor en si, en dicha línea hay que cambiar unos parámetros. En el código 10 se puede ver los cambios a realizar. Si el parámetro `debug` está en `True` el servidor se reinicia si se realiza un cambio en el código, puesto que el código ya no se va a modificar, se establece dicho parámetro en `False`. Establecer el parámetro `host` a 0.0.0.0 indica que el servidor se ejecutará en la IP pública de la máquina. Si esto no se indica, el servidor sólo es accesible desde localhost.

```
app.run(debug=True)

app.run(debug=False, host='0.0.0.0')
```

Código 10: Antes y después de la línea que hay que modificar en la aplicación

Realizado este cambio ya podemos lanzar la instancia EC2 de Amazon. Para ello hay que entrar en https://aws.amazon.com/es/ crear una cuenta, asociar una tarjeta para que se puedan cobrar posibles gastos y, seguidamente, crear una instancia EC2. En el

momento de crear la instancia se puede elegir entre varios planes, para este proyecto se ha utilizado el plan t2.micro ya que es el gratuito y sistema operativo Ubuntu ofrecido por Amazon. Mientras se crea dicha instancia llega un momento en el que se pregunta los accesos que se quieren establecer en la máquina, se ha de establecer que se permita el tráfico HTTP (puerto 80) y el tráfico SSH (puerto 22) desde cualquier máquina. Una vez creada la máquina y antes de lanzarla por primera vez se dará al usuario la posibilidad de descargar una clave. Dicha clave es la que permitirá al usuario establecer conexión SSH con el servidor, esta clave no se puede perder ya que no hay otra oportunidad para volver a generarla. Para establecer conexión al servidor se ha de ejecutar en la terminal el comando del código 11.

```
1 ssh -i <path to .pem file> ubuntu@<public DNS address>
```

Código 11: Comando para entrar a la máquina virtual de Amazon EC2 a través de SSH

Seguidamente se accederá a la máquina y ya se podrá preparar el despliegue del servidor. El primer paso es descargar todo lo necesario para ejecutar el servidor: Flask, Python 3.4, Python3-pip, PyMySQL, mysql-server, entre otros y también el fichero requirements.txt que se adjunta en el servidor. Posteriormente, se ha de descargar todo lo necesario para su despliegue en Apache tal y como: Apache2, Apache2-utils, git (para realizar un clone del repositorio del servidor), mod wsgi, etc.

Una vez descargado todo lo necesario, se reinicia Apache2 y en el directorio `/var/www/` se realiza un `git clone` del repositorio del servidor. Dicho repositorio incluye ya el fichero .wsgi necesario para que apache ejecute el servidor. El nombre del fichero .wsgi ha de ser el mismo que el fichero .py con el código del servidor. El contenido del fichero `serverAPI.wsgi` es el del código 12.

```
import sys
sys.path.insert(0, '/var/www/serverAPI')

from serverAPI import app as application
```

Código 12: Contenido del fichero serverAPI.wsgi

Seguidamente, hay que dirigirse al directorio /etc/apache2/sites-available/ y crear un nuevo fichero, el nombre es indiferente pero en este proyecto se ha creado el fichero `amazonaws.com.conf`. Este es el fichero que dice a Apache que hacer cada vez que alguien accede al servidor. El contenido del fichero es el del código 13.

```
<VirtualHost *:80>
        ServerName serverAPI
        WSGIScriptAlias / /var/www/serverAPI/serverAPI.wsgi
        WSGIPassAuthorization on
        <Directory /var/www/serverAPI/>
                Order allow,deny
                Allow from all
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel info
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Código 13: Contenido del fichero amazonaws.com.conf

Establecer el campo WSGIPassAuthorization a `on` es muy importante, sin este campo apache no soporta cabeceras HTTP con autentificación.

Por último, sólo hay que activar el fichero configurado, desactivar el por defecto y reiniciar apache con los comandos del código 14.

```
sudo a2ensite amazonaws.com
sudo a2dissite 000-default
```

Código 14: Comandos para activar un servicio y desactivar otro

7. Planificación temporal

7.1. Planificación general

7.1.1. Duración del proyecto

La duración estimada del proyecto es aproximadamente de unos 4 meses y medio: desde el 16 de febrero hasta el 29 de junio que empiezan los turnos de defensa de proyectos. La planificación del proyecto se ha realizado con el objetivo de finalizarlo el 31 de mayo para así poder testear la aplicación un par de semanas con el objetivo de mejorar la interfaz, experiencia de usuario y realizar todo lo referente a la documentación.

7.1.2. Recursos

Los recursos utilizados para este proyecto han sido:

- Un sólo individuo encargado de desarrollar y de ser el cliente de su propio proyecto
- Ordenador con Windows 8.1 y Ubuntu 12.04
- Máquina virtual Amazon EC2 (*Elastic Compute Cloud*) para ejecutar el servidor
- Android Studio, Adobe Photoshop CS6 Extended y PyCharm
- Git y Github para control de versiones

7.1.3. Plan de acción

En el final de esta sección se adjunta un diagrama de Gantt con la planificación de todas las tareas. En este diagrama aparecen de color naranja las tareas que en un principio se pensaron que podrían sufrir riesgos así acabaron siendo.

Estas tareas sufren de riesgos ya que se debe a que son los primeros contactos a la hora de utilizar el framework para desarrollar el servidor y a la hora de desarrollar la

aplicación Android. Una vez pasadas estas primeras etapas se adquirió una mayor soltura a la hora de desarrollar el resto de la aplicación, tanto por la parte cliente como por la del servidor. Además, a estas tareas que pueden llegar a ser conflictivas les fue destinado un tiempo mayor que al resto, para poder asimilar ciertos retrasos que puedan producirse.

7.1.4. Análisis de alternativas

Este proyecto se podría haber desarrollado con el simple objetivo de un tablón de anuncios de músicos buscando grupo o de grupos buscando un componente en concreto. Sin embargo, yo buscaba un servicio distinto: un servicio que además de lo anterior también ayude a la evolución de los grupos y músicos, además de ser una herramienta para que los fans de la música tengan una agenda de eventos.

Para empezar, hay que hablar sobre la plataforma. Tal y como se ha dicho anteriormente la aplicación cliente es Android. Me he decidido por Android y no por una página web por dos principales razones: no hay ningún servicio similar todavia existente en dicha plataforma y por el uso en aumento de los smartphones debido a que muchos usuarios incluso dejan de utilizar sus ordenadores porque sus necesidades ya son cubiertas por los smartphones.

Posteriormente, cabe destacar que es necesario disponer de distintos tipos de cuenta: fan, músico, grupo y propietario de local. Esto se podría haber hecho creando una cuenta de un determinado tipo e ingresar en la aplicación con el típico usuario y contraseña. Sin embargo esta idea no me acabó de convencer y me decidí por crear una cuenta única por usuario, dicha cuenta puede estar asociada a un perfil de fan o artista (exclusivamente una de las dos) y/o propietario de local. A su vez, un perfil de artista puede estar asociado a más de una banda y viceversa. Con este sistema de perfiles por cuenta tenemos una mecánica de aplicación más ágil y dinámica, reduciendo al mínimo el número de veces que el usuario ha de introducir contraseñas.

Una vez decididos que tipos de perfiles poblarán esta aplicación hubo que decidir que capacidades tendría cada uno. En un principio había muchas alternativas: que los fans sólo tuvieran capacidad para ver el calendario de eventos, que además pudieran seguir a sus músicos y grupos favoritos o que pudieran enviar mensajes a otros perfiles. Lo mismo ocurrió con el diseño de capacidades de otros perfiles. Tras muchas decisiones llegué al diseño actual:

- **Fan**: Este tipo de perfil puede seguir a los otros 3 tipos para ver sus publicaciones, ver en que eventos participan sus grupos favoritos, o cuales son los eventos que crean sus locales más cercanos. Sin embargo no pueden enviar mensajes a otros perfiles para evitar que perfiles con fama sufran spam de fans.

- **Artista**: Este perfil puede hacer todo lo que puede hacer el de fan además de poder enviar mensajes a otros perfiles de artista y de grupo pero no a propietarios de local (ya que los propietarios de locales solo invitarán a grupos) y puede publicar mensajes en su propio perfil o en el *feed* de sus seguidores. Además, este perfil también tiene la capacidad de crear un nuevo tipo de perfil: el de grupo, el cual en el momento de creación el único componente será él mismo.

- **Grupo**: Este perfil no es de uso único como puede serlo el de fan o artista, si no que es compartido entre todos los componentes del grupo, por lo que no tiene la funcionalidad de seguir a otros perfiles pero si la de enviar mensajes a artistas, otros grupos y a propietarios de local, capacidad para crear publicaciones para informar a sus fans de sus noticias, etc. Además, este tipo de perfil puede invitar a otros artistas a unirse al grupo.

- **Propietario de local**: Este perfil tampoco es de uso único, por lo que sus capacidades serán las de invitar a cuentas (por correo electrónico) a poder entrar y administrar el perfil, contactar con grupos a través de mensajes, crear eventos e invitar grupos a dichos eventos.

7.2. Descripción de las tareas

Todas las tareas descritas a continuación incluyen un testeo propio e individual sobre ellas mismas, ya sea haciendo requests HTTP para testear la API o compilaciones y ejecuciones del código de la aplicación Android.

7.2.1. Investigación y toma de decisiones globales

En esta tarea se incluye todo el tiempo dedicado a investigar aplicaciones similares y pensar funcionalidades que hagan el proyecto destacar sobre estas. Del mismo modo, hay que investigar que framework puede ser el más adecuado para el proyecto: que ofrezca potencia y capacidad pero al mismo tiempo facilidad de uso y aprendizaje.

7.2.2. Desarrollo del servidor

- **Diseño de la base de datos y llamadas de la API**: Esta tarea incluye el diseño del modelo entidad-relación de la base de datos y la especificación de todas las funciones que debe tener la API del servidor.
- **Cuentas, perfiles asociados a cada cuenta y autentificación**: Desarrollo de las funciones de la API referentes a crear cuentas, modificarlas, borrarlas y conseguir información sobre ellas tal y como los perfiles que hay creados por cada cuenta (artista, banda, fan, propietario de local). Además de autentificación por contraseña en el log-in de cuenta y con token en las posteriores.
- **Sistema de seguimiento de perfiles**: Tarea que incluye el desarrollo del sistema de seguidores: poder seguir a un perfil, dejar de seguirlo, poder ver los seguidores de un perfil y los seguimientos de un determinado perfil.
- **Sistema de miembros por bandas, instrumentos y eventos**: Desarrollo del sistema que indica los miembros que hay en cada grupo, los grupos en los que es componente un artista, que instrumentos toca un artista, que miembros (correspondientes a instrumentos) quedan en un determinado grupo para estar completo

y toda la gestión de eventos tal y como crearlos (desde un perfil de propietario de local) e invitar a grupos.

- **Sistema de publicaciones y mensajes entre perfiles**: Esta tarea incluye el desarrollo de publicaciones por cada perfil, es decir, cada perfil tiene la capacidad de escribir publicaciones que serán visibles desde su perfil y desde un *feed* en la pantalla inicial de sus seguidores. Estas publicaciones contienen novedades sobre sus actividades, noticias, etc. Además, también se incluye el desarrollo de mensajes entre bandas y artistas y entre dueños de local y bandas. El sistema de mensajería es similar a un sistema de correo electrónico pero interior en la aplicación.

- **Sistema de galería multimedia de cada perfil**: Los perfiles de bandas y artistas tienen la capacidad de poder adjuntar videos de YouTube además de asociar un perfil de SoundCloud a sus perfiles con el objetivo de que cualquier usuario que visite dichos perfiles pueda ver a dicho artista o grupo en acción, sus habilidades, estilo, etc. Esta tarea también el incluye la capacidad de subir fotos de perfil al servidor y de transferirlas del mismo al cliente.

- **Sistema de búsqueda de perfiles y eventos**: Esta tarea incluye el desarrollo del sistema de búsqueda de distintos tipos de perfiles con distintos campos de búsqueda, ya sea nombre, localización, instrumento que toca, género, etc.

7.2.3. Desarrollo del cliente

- **Diseño general del funcionamiento de la aplicación**: Esta tarea corresponde al diseño en papel de los distintos layouts que tendrá la aplicación Android, la función de cada botón, las interacciones entre cada pantalla, etc.

- **Creación y acceso de cuentas y perfiles**: Esta tarea incluye el desarrollo, diseño de los layouts y los requests HTTP referentes a crear, modificar, eliminar y visualizar cuentas y perfiles.

- **Formulario de búsqueda y visualización de resultados**: Incluye el desarrollo,

diseño de los layouts y los requests HTTP referentes a buscar perfiles y mostrar resultados de búsquedas.

- **Seguir perfiles y visualizar seguidores/seguimiento**: Incluye el desarrollo, diseño de los layouts y los requests HTTP referentes a seguir perfiles y la visualización de los seguidores y seguimientos de cada perfil.

- **Publicaciones, miembros en banda y viceversa, miembros restantes**: Incluye el desarrollo, diseño de los layouts y los requests HTTP referentes a crear publicaciones en cada perfil, visualizarlas y modificarlas y eliminarlas si se es el autor. Además de todo lo referente a crear grupos desde un perfil de artista, invitar a otros artistas a unirse, visualizar las bandas de artista y viceversa.

- **Creación y visualización de ficheros multimedia de cada perfil y mensajería**: Incluye el desarrollo, diseño de los layouts y los requests HTTP referentes a la creación y visualización de contenido multimedia. Este contenido se resume a las fotos de perfil, vídeos de YouTube y perfiles asociados de SoundCloud. Además también se incluye la pantalla de mensajes recibidos ordenados por tipo de perfil,

- **Creción de eventos, eventos por banda y sala, calendario de eventos y GPS**: Incluye el desarrollo, diseño de los layouts y los requests HTTP referentes a la creación de eventos desde un perfil de propietario de local así como su modificación y eliminación. Esta tarea también incluye la visualización de en que eventos participa un grupo y viceversa, así como un calendario de eventos accesible por cualquier tipo de perfil y de un mapa que muestra los eventos cercanos a la ubicación GPS.

7.3. Cambios desde la planificación inicial

En esta sección se discuten los cambios que han tenido lugar entre la planificación inicial y el desarrollo que finalmente ha tenido lugar.

Empezando por el desarrollo del servidor, realmente no hubo muchos cambios. Todas las funcionalidades y su duracion se han llevado a cabo en el mismo orden y duración. Sin embargo, ha hecho falta modificar algunas llamadas ya existentes o añadir nuevas durante el desarrollo del cliente.

En cuanto el desarrollo del cliente ha habido más cambios, tanto en el orden de las tareas como su duración e incluso las funcionalidades implementadas. Por ejemplo, inicialmente se planeó que se desarrollaría antes la visualizacion de qué miembros hay en un grupo y viceversa que el sistema de seguimiento de perfiles pero este último, al final, se ha desarrollado antes. De un modo similar, anteriormente había una tarea que incluía el desarrollo de publicaciones y de mensajería. Estas dos partes han sido dispersadas: las publicaciones se unieron con el desarrollo miembros en grupos y viceversa (como se puede observar en el listado anterior) y la mensajeria con los ficheros multimedia.

Tal y como se ha dicho al inicio de esta sección, la planificación se realizó para que el desarrollo se acabase el 31 de mayo. Por lo que del 1 de junio al 14 se podría testear la aplicación, arreglar bugs y mejorarla y del 15 al 28 preparar la defensa y también la memoria. La planificación de este mes se ha visto alterada ya que realmente no había hasta el 28 para redactar la memoria, así que la primera semana de junio ha sido para testeo, arreglo de bugs y mejoras, la segunda y tercera para la redacción de la memoria y la última, si es necesario, para más mejoras y, sobretodo, preparación de la defensa. La nueva planificación se adjunta en el siguiente apartado.

7.4. Diagrama de Gantt

	Nombre	Duración	Inicio	Fin
1	Investigación y toma de decisiones globales	20h	09/02/2015	13/02/2015
2	**GEP**	32.31d	**18/02/2015**	**22/03/2015**
3	Alcance	9.25h	18/02/2015	20/02/2015
4	Planificación temporal	8.25h	23/02/2015	25/02/2015
5	Gestión económica y sostenibilidad	9.25h	01/03/2015	03/03/2015
6	Presentación preliminar	6.25h	07/03/2015	08/03/2015
7	Contextualización y bibliografía	15.25h	12/03/2015	15/03/2015
8	Pliego de condiciones	8h	16/03/2015	17/03/2015
9	Presentación oral y documento final	18.25h	18/03/2015	22/03/2015
10	**Desarrollo del servidor**	49d	**19/02/2015**	**08/04/2015**
11	Diseño de la base de datos y llamadas de la API	16h	19/02/2015	22/02/2015
12	Cuentas, perfiles asociados a cada cuenta y autentificación	22h	23/02/2015	28/02/2015
13	Sistema de seguimiento de perfiles	18h	28/02/2015	04/03/2015
14	Sistema de miembros por bandas, instrumentos y eventos	24h	05/03/2015	10/03/2015
15	Sistema de publicaciones y mensajes entre perfiles	40h	11/03/2015	20/03/2015
16	Sistema de galería multimedia de cada perfil	26h	21/03/2015	27/03/2015
17	Sistema de búsqueda de perfiles y eventos	48h	28/03/2015	08/04/2015
18	**Desarrollo del cliente**	53d	**09/04/2015**	**31/05/2015**
19	Diseño general del funcionamiento de la aplicación	4h	09/04/2015	09/04/2015
20	Creación y acceso de cuentas y perfiles	40h	10/04/2015	19/04/2015
21	Formulario de búsqueda y visualizacion de resultados	28h	20/04/2015	26/04/2015
22	Seguir perfiles y visualizar seguidores/seguimientos	36h	27/04/2015	05/05/2015
23	Publicaciones, miembros en banda y viceversa, miembros restantes	32h	06/05/2015	13/05/2015
24	Creación y visualización de ficheros multimedia de cada perfil y mensajería	40h	14/05/2015	23/05/2015
25	Creción de eventos, eventos por banda y sala, calendario de eventos y GPS	32h	24/05/2015	31/05/2015
26	Testeo, arreglo de bugs y mejoras	28h	01/06/2015	07/06/2015
27	Redacción memoria	56h	08/06/2015	21/06/2015
28	Preparación Defensa	28h	22/06/2015	28/06/2015

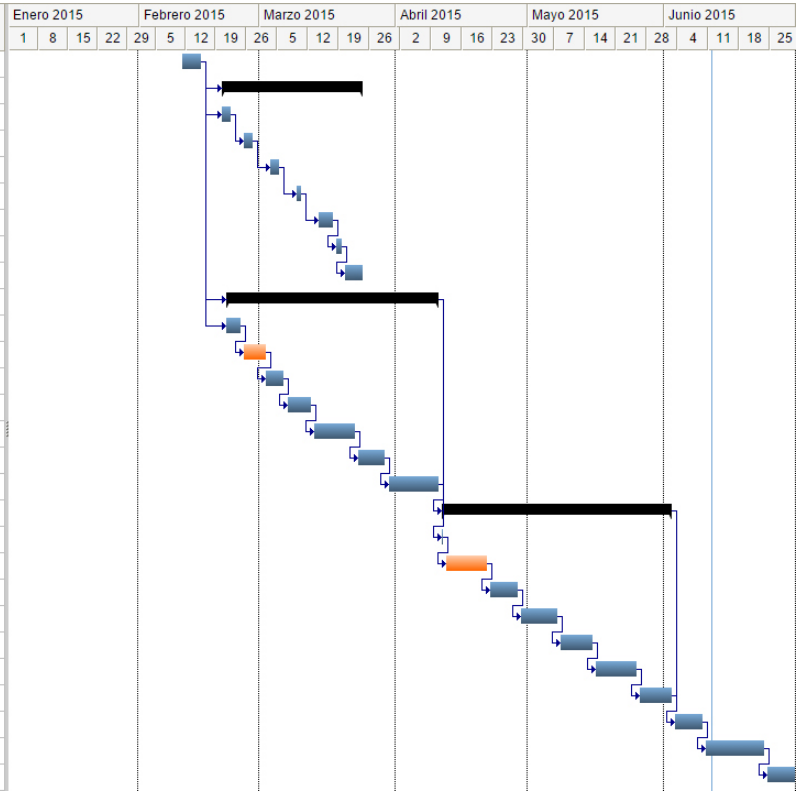

Figura 33: Diagrama de Gantt obtenido con la herramienta Gantter

8. Gestión económica

8.1. Consideraciones iniciales

Hace falta considerar que este es un proyecto universitario hecho por un solo alumno, por lo que este ha desarrollado todos los roles implicados en el proyecto y con un coste nulo. Aún así, los costes que se indicarán en este documento serían los equivalentes a un proyecto real realizado en empresa.

8.2. Identificación y estimación de los costes

8.2.1. Presupuesto de recursos humanos

En este proyecto han participado (distintos roles hechos por el mismo estudiante) un jefe de proyecto, un analista programador, un programador, un diseñador y un beta-tester (o varios). Como se puede observar, los salarios son 50€/hora, 40€/hora, 30€/hora, 25€/hora y 20€/hora, respectivamente. En la tabla 1 podemos ver los costes asociados a las tareas que ya han sido especificadas y planificadas.

Actividad	Horas	Rol	Precio	Coste
Investigación y toma de decisiones globales	20h	Jefe de proyecto	50€/h	1.000€
GEP				
Alcance	9,25h	Jefe de proyecto	50€/h	462,5€
Planificación temporal	8,25h	Jefe de proyecto	50€/h	412,5€
Gestión económica y sostenibilidad	9,25h	Jefe de proyecto	50€/h	462,5€
Presentación preliminar	6,25h	Jefe de proyecto	50€/h	312,5€
Contextualización y bibliografía	15,25h	Jefe de proyecto	50€/h	762,5€
Pliego de condiciones	8h	Jefe de proyecto	50€/h	400€
Presentación oral y documento,final	18,25h	Jefe de proyecto	50€/h	912,5€
Desarrollo del servidor				
Diseño de la base de datos y llamadas de la API	16h	Analista programador	40€/h	640€
Cuentas, perfiles asociados a cada cuenta y autentificación	22h	Programador	30€/h	660€
Sistema de seguimiento de perfiles	18h	Programador	30€/h	540€
Sistema de miembros por bandas, instrumentos y eventos	24h	Programador	30€/h	720€
Sistema de publicaciones y mensajes entre perfiles	40h	Programador	30€/h	1200€
Sistema de galería multimedia de cada perfil	26h	Programador	30€/h	780€
Sistema de búsqueda de perfiles y eventos	48h	Programador	30€/h	1440€
Desarrollo del cliente				
Diseño general del funcionamiento de la aplicación	4h	Analista programador	40€/h	160€
Creación y acceso de cuentas y perfiles	40h	Programador	30€/h	1200€
Formulario de búsqueda y visualización de resultados	28h	Programador	30€/h	840€
Seguir perfiles y visualizar seguidores/seguimiento	36h	Programador	30€/h	1080€
Publicaciones, miembros en banda y viceversa, miembros restantes	32h	Programador	30€/h	960€
Creación y visualización de ficheros multimedia de cada perfil y mensajería	40h	Programador	30€/h	1200€
Creción de eventos, eventos por banda y sala, calendario de eventos y GPS	32h	Programador	30€/h	960€
Diseño del cliente	20h	Diseñador	25€/h	500€
Testeo	14h	Beta-Tester	20€/h	280€
Mejoras	14h	Programador	30€/h	420€
Documentación	84h	Jefe de proyecto	50€/h	4200€
Total	**632,5h**			**22.505€**

Tabla 1: Presupuesto de recursos humanos

8.2.2. Presupuesto de hardware

En este proyecto se ha utilizado un solo ordenador además de una máquina virtual Amazon EC2 que aloja el servidor con el objetivo de distribuir la aplicación a los beta-testers. El ordenador de sobremesa tiene un precio de 700 y la máquina virtual en Amazon EC2 no tiene coste ya que sólo se utiliza el *free tier*.

Producto	Precio	Porcentaje de uso	Precio imputado al proyecto
Ordenador sobremesa	700€/ 4 años	632,5 horas	12,68€
Amazon EC2	0€/ mes	336 horas (dos semanas de testeo)	0€
Total			**12,68€**

Tabla 2: Presupuesto de hardware

8.2.3. Presupuesto de software

En cuanto al software utilizado en este proyecto se ha Adobe Photoshop CS6 Extended, el cual tiene un precio de 966, Android Studio, cuyo uso es gratuito y Pycharm, cuya

licencia son 99 (adquisición y primer año) y 59 cada renovación anual.

Producto	Precio	Porcentaje de uso	Precio imputado al proyecto
Adobe Photoshop CS6 Extended	966€/ 3 años	20 horas	0,74€
Android Studio	0€	-	0€
PyCharm	217€/ 3 años	431 horas	3,56€
Total			**4,30€**

Tabla 3: Presupuesto de software

8.2.4. Presupuesto de mantenimiento

Para mantener la aplicación activa hay que seguir pagando la máquina virtual de Amazon EC2. En el presupuesto de hardware solamente está contabilizado el precio de un solo mes, el correspondiente a testeo de la aplicación. Para prolongar este tiempo habrá que seguir pagando este servicio. Dado que hay muchas instancias disponibles y no se puede prever el futuro del proyecto, adjunto el enlace con todos los precios: http://aws.amazon.com/es/ec2/pricing/.

8.2.5. Presupuesto total

Finalmente, sumamos todos los tipos de presupuesto para saber el total de nuestro proyecto.

Concepto	Coste
Recursos humanos	22.505€
Recursos de hardware	12,68€
Recursos de software	4,30€
Total	**22.521,98€+ IVA = 27.251€**

Tabla 4: Presupuesto total

Con un coste total de 27.251 se concluye que es un proyecto competitivo en su sector y viable.

8.3. Control de gestión

Con el objetivo de controlar el tiempo real que se ha invertido en el proyecto se han ido actualizando las horas dedicadas a cada tarea. De esta forma se ha podido controlar

cual era el coste estimado y cual ha sido el real. Tal y como se ha dicho anteriormente, el coste real ha sido de 27.251€y el que se estimó en un principio era de 25.973€. Como se puede observar, el coste no ha variado mucho, por lo que se hizo una estimación correcta.

9. Sostenibilidad y compromiso social

Después de haber estimado el coste en cuanto a recursos humanos, hardware, software y haber especificado un control de gestión nos encontramos con el estudio de la sostenibilidad del proyecto en distintos medios. Primero mostramos la matriz de sostenibilidad del proyecto y posteriormente justificamos cada uno de los ámbitos.

	Sostenibilidad económica	Sostenibilidad social	Sostenibilidad ambiental	Total
Valoración	8	8	8	24

Tabla 5: Matriz de sostenibilidad

9.1. Sostenibilidad económica

Existe una avaluación de costes, tanto de recursos materiales como humanos, especificada anteriormente. Se ha tenido en cuenta los posibles contratiempos, de forma que se han aumentado las horas de trabajo en todas las tareas, pero sobretodo en las críticas. El coste del proyecto es viable ya que seguramente sería mucho más barato que el especificado ya que en un ámbito profesional llevaría muchas menos horas. Se podría realizar un proyecto similar en menos tiempo y por lo tanto menos coste, pero en el caso de este proyecto se ha de tener en cuenta que el estudiante va a aprender a usar las tecnologías mientras desarrolla con ellas. El tiempo dedicado a cada tarea está justificado, las tareas que tienen un mayor tiempo son tareas que conllevan nuevo aprendizaje. No hay prevista ningún tipo de colaboración con cualquier otro proyecto.

9.2. Sostenibilidad social

Este proyecto impacta en las dimensiones musicales de una ciudad o incluso de un país si una banda consigue viajar lejos de su ciudad gracias a este servicio. Además, hará la vida mucho más fácil al sector musical amateur (y no tan amateur) y no va a perjudicar a ningún otro sector de la música o social. Se prevé que los usuarios de esta aplicación mejoren su calidad de vida, ya sea obteniendo oportunidades que de otro

modo no podrían conseguir, ahorrando tiempo o incluso dinero al no tener que acudir a promotoras musicales.

9.3. Sostenibilidad ambiental

Los recursos necesarios durante las diferentes fases del proyecto no han cambiado, el único recurso utilizado será un ordenador y la electricidad que este consumirá. Por esa misma razón este proyecto tiene un impacto ambiental menor, solamente el relacionado al consumo eléctrico del ordenador utilizado para el desarrollo. Realizar la actividad que soluciona este proyecto sin existir este servicio no conlleva ningún tipo de consumo de papel o energía ya que actualmente todas estas comunicaciones se realizan vía Internet. Tampoco se requiere energía para realizar este proyecto ya que sólo se utiliza un ordenador para desarrollar el software necesario. En el proyecto no se realiza ningun prototipo físico de ningún dispositivo, por lo que no hay que tener en cuenta aspectos de reciclaje o fabricación

10. Conclusiones

10.1. Logro de objetivos

Una vez finalizado el desarrollo de este proyecto se ha obtenido el producto que se deseaba. Esta red social sirve como herramienta comunicativa entre fans, artistas, grupos y propietarios de locales pero además también como aplicación de entretenimiento y estilo de vida.

Gracias a este servicio artistas pueden encontrar a otros artistas para crear un grupo, o simplemente para conocerse entre ellos, compartir experiencias y música. Los artistas también pueden encontrar a grupos que busquen componentes y viceversa. Por otro lado, los propietarios de locales pueden crear eventos e invitar a cualquier grupo que utlice este servicio a participar al evento. Además, gracias a la creación de estos eventos los usuarios disponen de un mapa con eventos cercanos a la ubicación GPS y también de un calendario de eventos. Todo esto acompañado de una interfaz simple pero agradable, intuitiva y que responda a las interacciones del usuario, que facilita la navegación entre perfiles, mensajes, eventos, publicaciones, etc. lo máximo posible y aportando al usuario el máximo de información en todo momento.

En cuanto a la planificación temporal, tal y como se ha dicho en su sección ha sufrido varios cambios respecto a la que se planificó en el inicio del proyecto. En la parte del desarrollo del servidor no ha habido cambios, pero por el contrario, en la parte del cliente sí. Se ha cambiado el orden de algunas tareas e incluso se ha cambiado algunas de las funcionalidades implementadas en estas.

10.2. Futuro trabajo

Aunque se ha cumplido con todos los objetivos propuestos en un inicio y la aplicación es usable, no se trata de un producto final y está sujeto a cambios. Al ser una red social un papel muy importante para su mejora es escuchar la opinión de los usuarios para

saber que funcionalidades extras necesitan. Estos cambios, por supuesto, derivarán en cambios tanto de la aplicación cliente como la aplicación servidor.

Sin embargo, si que hay unas cuantas mejoras que se pueden realizar sin necesitar el *feedback* de los usuarios. En cuanto a implementación, hay mejoras que se pueden realizar tal y como un mejor diseño de base de datos optimizando así el número de modelos necesarios, sobretodo en el caso de los mensajes y de las publicaciones. También se pueden optimizar algunas de las peticiones que se realizan desde la aplicación cliente con el objetivo de minimizar el número de peticiones realizadas al servidor.

En cuanto a funcionalidades, para los perfiles de artistas y grupos, se podría crear una nueva pestaña con el titulo de "enlaces de interés" y que el usuario pudiera adjuntar diferentes enlaces como por ejemplo un enlace a una tienda con *merchandising* del grupo, enlace a Bandcamp, etc. Por último, también se podría añadir un quinto perfil: promotora. Las promotoras tienen un gran papel e importancia en el mundo de la música, se encargan de organizar conciertos, buscar grupos, dar grupos a conocer, etc.

10.3. Valoración personal

El desarrollo de este proyecto realmente me ha servido de mucho tanto profesionalmente como personalmente. Ha sido la primera vez que he desarrollado un proyecto de este tamaño y desde cero, he aprendido nuevos lenguajes y frameworks y he profundizado en Android ya que sólamente sabía programar cosas muy básicas. A lo largo del proyecto he agradecido haber escogido una metodología ágil ya que me ha ayudado mucho a la hora de seguir el control y el tiempo sobre el desarrollo. He tenido que tomar decisiones tanto para añadir funcionalidades como para eliminar otras por falta de tiempo. También he aprendido a planificar un proyecto con tanto detalle: presupuesto, gestión económica y sostenibilidad, metodología, planificación temporal, etc. y he hecho cosas que nunca había hecho como crear una máquina virtual en Amazon EC2 y utilizar Apache para ejecutar el servidor. Personalmente, me hubiera gustado haber tenido más tiempo pa-

ra desarrollar el proyecto por varios motivos: para añadirle más funcionalidades y por haber hecho así que el desarrollo de este proyecto hubiera sido una afición para mí. Lo que quiero decir con esto último es que, aunque he disfrutado mucho desarrollando esto, conforme se ha ido acercando la fecha de entrega he ido notando una presión que me impedía seguir disfrutando de la misma forma que como al principio. Aún así, no me quejo de la falta de tiempo que ha sido derivada del aprendizaje casi desde cero y sé que esto es lo que me encontraré en un entorno profesional.

11. Referencias

[1] Wikipedia, "Application Programming Interface." [Online]. Available: http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

[2] ——, "Base de datos relacional." [Online]. Available: http://es.wikipedia.org/wiki/Base_de_datos_relacional

[3] ——, "Cliente." [Online]. Available: http://es.wikipedia.org/wiki/Cliente_%28inform%C3%A1tica%29

[4] Flask, "Flask." [Online]. Available: http://flask.pocoo.org/

[5] Wikipedia, "Framework." [Online]. Available: http://es.wikipedia.org/wiki/Framework

[6] ——, "Hipermedia." [Online]. Available: http://es.wikipedia.org/wiki/Hipermedia

[7] ——, "Hypertext Transfer Protocol." [Online]. Available: http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[8] ——, "Integrated Development Interface." [Online]. Available: http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

[9] Jinja2, "Jinja2." [Online]. Available: http://jinja.pocoo.org/docs/dev/

[10] Wikipedia, "JavaScript Object Notation." [Online]. Available: http://es.wikipedia.org/wiki/JSON

[11] ——, "MySQL." [Online]. Available: http://es.wikipedia.org/wiki/MySQL

[12] ——, "Python." [Online]. Available: http://es.wikipedia.org/wiki/Python

[13] ——, "Representational State Transfer." [Online]. Available: http://es.wikipedia.org/wiki/Representational_State_Transfer

[14] ——, "Servidor Web." [Online]. Available: http://es.wikipedia.org/wiki/Servidor_web

[15] SoundCloud, "SoundCloud." [Online]. Available: https://soundcloud.com/

[16] Wikipedia, "Template processor." [Online]. Available: http://en.wikipedia.org/wiki/Template_processor

[17] Werkzeug, "Werkzeug." [Online]. Available: http://werkzeug.pocoo.org/

[18] Wikipedia, "WSGI." [Online]. Available: http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

[19] W3Techs, "Usage Statistics and Market Share of Server-side Programming Languages for Websites." [Online]. Available: http://w3techs.com/technologies/overview/programming_language/all

[20] Wikipedia, "Server-side scripting." [Online]. Available: http://en.wikipedia.org/wiki/Server-side_scripting

[21] R. Brown, "Django vs Flask vs Pyramid." [Online]. Available: https://www.airpair.com/python/posts/django-flask-pyramid

[22] M. Grinberg, *Flask Web Development*, M. Blanchette and R. Roumeliotis, Eds. O'Reilly, 2014.

[23] O'Reilly, "O'Reilly Media." [Online]. Available: http://www.oreilly.com/

[24] M. Grinberg, "Building Web APIs with Flask." [Online]. Available: http://shop.oreilly.com/product/0636920034803.do?cmp=af-prog-books-videos-product_cj_9781491911938_%25zp

[25] ——, "Miguel Grinberg Blog." [Online]. Available: http://blog.miguelgrinberg.com/

[26] Buscogrupo, "Busco grupo." [Online]. Available: http://buscogrupo.net/

[27] Bandmix, "Bandmix España." [Online]. Available: http://www.bandmix.es/

[28] Localrock, “Local Rock.” [Online]. Available: http://www.localrock.es/

[29] Joinmyband, “Join my band.” [Online]. Available: http://www.joinmyband.co.uk/

[30] Bandfinder, “Band finder.” [Online]. Available: http://www.bandfinder.com/

[31] Findamusician, “Find a musician.” [Online]. Available: http://www.find-a-musician.com/

[32] S. Alchemy, “SQL Alchemy.” [Online]. Available: http://www.sqlalchemy.org/

[33] Flask, “Flask-SQLAlchemy.” [Online]. Available: https://pythonhosted.org/Flask-SQLAlchemy/

[34] ComTech, “Kantar World Panel.” [Online]. Available: http://www.kantarworldpanel.com/global/smartphone-os-market-share/

[35] A. Query, “AQuery.” [Online]. Available: https://code.google.com/p/android-query/

[36] Wikipedia, “Model-view-controller.” [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[37] Amazon, “Amazon Web Services.” [Online]. Available: https://aws.amazon.com/es/